

BPM VENTILATION WITH OXIMETER

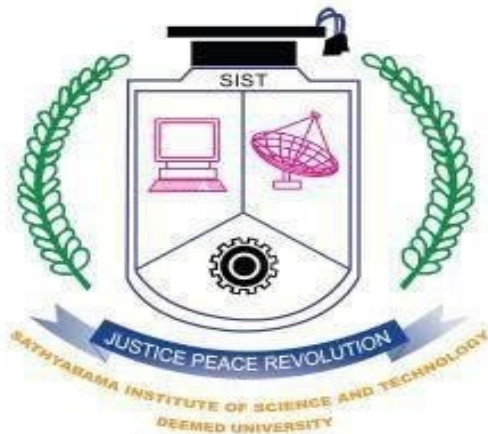
Submitted in partial fulfilment of the requirements for the award of

Bachelor of Engineering Degree in
Electronics and Communication Engineering

By

YARRAMSETTY GANESH (38130245)

VASIREDDY KARTHIK (38130233)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION

ENGINEERING

SCHOOL OF ELECTRICAL AND ELECTRONICS

ENGINEERING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MAY 2022



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **YARRAMSETTY GANESH** (38130245) and **VASIREDDY KARTHIK** (38130233) who carried out the project entitled "**BPM VENTILATION WITH OXIMETER**" under our supervision from December 2021 to May 2022.

Internal Guide

Dr. S. POORNAPUSHPAKALA M.E,Ph.D

Head of the Department

Dr. T. RAVI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

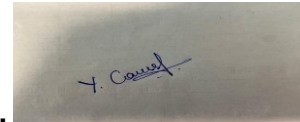
Internal Examiner

External Examiner

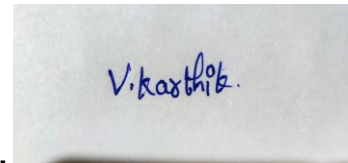
DECLARATION

We **YARRAMSETTY GANESH** (38130245) and **VASIREDDY KARTHIK** (38130233) hereby declare that the Project Report entitled “**BPM VENTILATION WITH OXIMETER**” done by us under the guidance of **Dr.S.Poornapushpakala** at Sathyabama Institute of Science and Technology, Chennai- 600119 is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

1.

A rectangular box containing a handwritten signature in blue ink that reads "Y. Ganesh".

2.

A rectangular box containing a handwritten signature in blue ink that reads "V. Karthik".

SIGNATURE OF CANDIDATES

DATE :

PLACE :

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr.N.M.Nandhitha, Dean, School of Electrical and Electronics Engineering** and **Dr.T.Ravi, Head of the Department, Dept. of Electrical and Electronics Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would also like to express our sincere and deep sense of gratitude to our internal guide **Dr.S.Poornapuspakala** for her valuable guidance and suggestions during the progressive reviews for our project.

ABSTRACT:

Human lungs use the reverse pressure generated by contraction motion of the diaphragm to suck in air for breathing. A contradictory motion is used by a ventilator to inflate the lungs by pumping type motion.

A ventilator mechanism must be able to deliver in the range of 10 – 30 breaths per minute, with the ability to adjust rising increments in sets of 2. Along with this the ventilator must have the ability to adjust the air volume pushed into lungs in each breath. Also, there is a setting to adjust the time duration for inhalation to exhalation ratio.

Apart from this the ventilator must be able to monitor the patients' blood oxygen level and exhaled lung pressure to avoid over/under air pressure simultaneously.

The ventilator we here design and develop using NODEMCU encompasses all these requirements to develop a reliable yet affordable DIY ventilator to help in times of pandemic.

We here use a silicon ventilator bag coupled driven by DC motors with 2 side push mechanism to push the ventilator bag. We use toggle switch from cloud switching and a variable pot to adjust the breath length and the BPM value for the patient.

Our system makes use of blood oxygen sensor along with sensitive pressure sensor to monitor the necessary vitals of the patient. An emergency buzzer alert is fitted in the system to sound an alert as soon as any anomaly is detected.

The entire system is driven by NODEMCU controller to achieve desired results and to assist patients in covid pandemic and other emergency situations.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
2	LITERATURE SURVEY	2
3	AIM AND SCOPE OF PROJECT	3
	3.1 AIM OF PROJCT	4
4	EXPERMETNTAL SETUP AND IMPLEMENTATION OF PROJECT	5
	4.1 EMBEDDED SYSTEMS	5
	4.1.1 OVERVIEW OF EMBEDDED SYSTEMS	5
	4.1.2 BLOCK DIAGRAM OF EMBEDDED SYSTEM	6
	4.1.3 CHARACTERISTICS OF EMBEDDED SYSTEM	7
	4.1.4 HETEROGENEOUS ARCHITECTURE	8
	4.1.5 SMALL AND LOW WIGHT	10
	4.2 UBIDOTS EXPLORER	11
	4.2 INTRODUCTION	11
	4.2.1 DEVICES	14
	4.2.2 VARIABLES	15
	4.2.3 SYNTHETIC VARIABLE ENGINE	16
	4.2.4 DASHBOARDS AND VISUALIZATIONS	17
	4.2.5 EVENTS	18
	4.3 NODEMCU	20
	4.3.1 INTRODUCTION	21
	4.3.2 NODEMCU DEVELOPMENT BOARD	22
	4.3.3 NODEMCU WITH ARDUINO IDE	23

4.4 PULSE SENSOR	24
4.4.2 HARDWARE OVERVIEW	25
4.4.3 SPECIFICATION	26
4.4.4 PULSE SENSOR PINOUT	28
4.4.5 SENSOR WIRING WITH ARDUINO	29
4.4.6 HEART BEAT PLOTTING	30
4.5 SPO2 SENSOR	31
4.5.1 WORKING OF SENSOR	32
4.5.2 INTERFACING OF SENSOR WITH ARDUINO	32
4.6 SERVO MOTOR	34
4.7 BUZZER	35
4.7.1 BUZZER PIN CONFIGURATION	36
4.7.2 WORKING PRINCIPLE	37
4.7.3 APPLICATIONS	38
4.8 BMP 180 SENSOR	38
4.8.1 SENSOR PINOUT	39
4.8.2 SENSOR SPECIFICATION	39
5 RESULT AND CONCLUSION	40
6 SUMMARY AND CONCLUSION	42
REFERENCE	42

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
4.1.2	BLOCK DIAGRAM OF EMBEDDED SYSTEM	7
4.1.4	EMBEDDED SYSTEM ARCHITECTURE	9
4.2.1	DATA HIERARCHY	11
4.21	UBIDOTS TIME SERIES	13
4.22	CREATING DEVICES	15
4.2.2	ADDING VARIABLES	16
4.2.3	SYNTHETIC VARIABLE	17
4.2.4	DIFF BTW DEFAULT AND SYNTHETIC VARIABLE	17
4.2.5	EVENTS	18
4.2.6	GEOFENCE	19
4.3.1	NODEMCU IMG	21
4.3.2	BACK VIEW OF NODEMCU	22
4.4.1	PULSE SENSOR	25
4.4.2	PULSE SENSOR OVERVIEW	25
4.4.3	PULSE SENSOR HARDWARE	26
4.4.4	WORKING OF PHOTO SENSOR	27
4.4.5	MEASURING OF READINGS	27
4.4.6	PULSE SENSOR PIN SPECIFICATION	28
4.4.7	PULSE SENSOR WIRING	29
4.4.8	GRAPHICAL REPRESENTATION OF READINGS	30
4.4.9	READINGD OF HEARTBEAT	31
4.5.1	SPO2 SENSOR	31
4.5.2	CIRCUIT CONNECTION	32
4.5.3	CONNECTIING WIRES ON BREADBOARD	33
4.5.4	OUTPUT READINGS OF SPO2 SENSOR	33
4.6.1	SERVO MOTOR	35
4.7.1	BUZZER	36
4.8.1	BMP PRESSURE SENSOR	39
5.1	UBIDOTS RESULT	40
5.2	HEART RATE VALUES	40
5.3	OXIMETER VALUES	41
5.4	PRESSURE VALUES	41

CHAPTER 1

INTRODUCTION

The health care sensors are playing an essential role in hospitals. The patient monitoring system is one of the major developments because of its innovative technology. An automatic wireless health monitoring system is used to measure patient's body temperature and heartbeat by using embedded technology.

The proposed system uses both the sensors like heartbeat sensor and temperature sensor. These sensors mainly involved in monitoring the condition of the patient.

But the sensor data cannot be upload to cloud. So each and every time doctor will go there and check the parameter. In some conditions doctor may busy means they never monitoring the person.

The general objective of this work was to produce a low-cost oximeter for better health monitoring, and which would also serve as an aid during periods of COVID-19.

The developed solution proved to be efficient in reading questions, capable of demonstrating information of: number of patients, number of readings, measurement date and time, monitoring by device, evolution in the last 7 days, cities with the highest index (for state or national analyzes), patient age, oximetry, BPM and patient concentration heat maps. The solution proves to be affordable in economic term.

CHAPTER 2

LITERATURE SURVEY

Rupali Shinde (2018), In this paper, economical and wearable pulse oximeter device using sparkfun oximeter sensor and raspberry pi v2 model B is presented. Nowadays whole world is facing COVID-19 pandemic from December 2019 and in corona patient blood oxygen level is goes down due to attack of virus on lungs. In this scenario economical and easy to use pulse oximeter is life saving device. To control pandemic; early detection of patients are important. We have presented easy to develop oximeter in this paper. We have connected sensor to Arduino uno using I2C protocol and using serial communication data is sent do raspberry pi using USB serial communication. We have achieved over 90% accuracy when result are compare with commercial health band. We also did comparative study with other sensor and we found that sparkfun sensor gives more accurate and quick results. Data visualization is implemented using python library.

A. Azhari (2019), A patch-type wireless wearable pulse oximeter system has been developed to measure the heart beat rate and oxygen saturation of blood (SpO_2) in reflective mode from a person's forehead. The system uses light sources of two wavelengths (red, 625 nm; infrared, 865 nm) in an optical sensor, separates photodetector signal and transimpedance amplifier output into infrared (IR) and red photoplethysmographic (PPG) signals, and then digitizes and transmits data wirelessly via a Bluetooth module to a remote PC in real time, where the SpO_2 value is calculated. The optical sensor, mounted on a 7 cm × 2 cm flexible sheet, is interfaced with an 8.5 cm × 3.5 cm wireless SpO_2 circuit board. The entire sensor and wireless SpO_2 system, which weighs only 15 g, is readily wearable on the forehead, unlike conventional non-wearable wired forehead pulse oximeters. Using our system, the reduction of SpO_2 value was observed from 100% to 89%, under an 80 s breathhold condition, which matched with the simultaneously measured SpO_2 values obtained using commercial pulse oximeters.

Vandana Millind Rohokale, Neeli Rashmi Prasad, and Ramji Prasad (2011), detailed that the beat oximeter changed the capacity to control oxygen constantly, totally, and harmlessly. Albeit utilized all over, it is our viewpoint and affirmed by research from many specialist organizations who don't have the foggiest idea about the essential standards of execution.

Joseph Bailey, Michael Fecto, and Noah L. Pandleton (2015), revealed that wire oximeters were utilized by specialists in emergency clinics and other wellbeing places to peruse SpO2 * levels and patients. They are generally utilized related to ECG sensors and other observing gadgets to give data and heartbeat oximeter readings to the patient's clinical record. The primary remote "oximeter" of the oximeter is independent and in the medical clinic, so the patient shouldn't be firmly checked.

Junaid Mohammed, Abhinav Takral, Adrian Philip Okneanu, Colin Jones, Chung-Horng Lung and Andy Adler(2014) are vital for COVID-19 patients. This study distinguished the IoT-based framework as a genuine checking program in view of estimating the patient's internal heat level, pulse, and oxygen levels, which are key measures for genuine treatment.

Tae-Yun Kim, Sungkwan Youm, Jai-Jin Jung, and Eui-Jik Kim (2015), show that the undertaking carries out a mechanically wearable, lightweight, low-power, remote-based bio-signal control framework. . This execution is an interesting advance in wellbeing observing. The quantity of individuals needing clinical consideration is expanding step by step, and conventional bio-signal observing frameworks require hospitalization.

Carsten Aalders and Charles R. Abbruscato (2017), Instructions to enact dial and play cooperation. It shows the utilization of specific words, constructions, and ways of behaving in a telehealth climate that limits choice as an essential method for correspondence. The norms characterize the overall capacity of telemetry beat oximeter execution.

CHAPTER 3

AIM AND SCOPE OF THE PROJECT

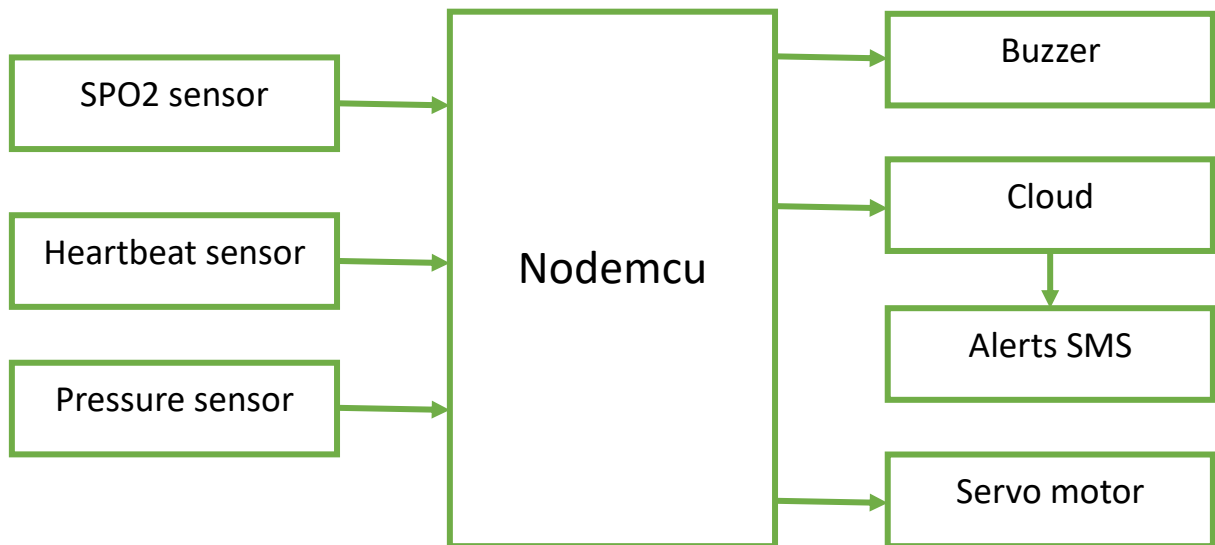
The human lung utilizes a lot of speed to inhale air because of the limiting of the stomach. The other way utilizes a fan to expand the kind of pressure lungs. The respiratory framework ought to have the option to inhale 10 to 30 times each moment and change the volume in stage 2. To wrap things up is the setting to change the endlessly breathing time. Likewise, the air should have the option to control the patient's blood oxygen level and aspiratory pulse to stay away from a similar gaseous tension.

The breezes we convert here and foster utilizing Nodemcu contain these prerequisites to assemble a dependable however reasonable DIY to assist with a plague. Here we utilize two strategies for pushing the side to push the silicon airbag and sack associated with the DC engine. We use cloud-based collaborations and factors to change the profound tallness and worth of BPM in patients. Our framework utilizes blood sensors and sensors to screen patient requirements. The framework is additionally informed to hear a crisis call. To accomplish the ideal outcomes, the whole framework is overseen by a NODEMCU regulator, which helps patients during the spread of COVID and different crises.

3.1 AIM OF THE PROJECT:

- The main aim of the proposed work is to produce a low-cost oximeter for better health monitoring, and which would also serve as an aid during periods of COVID-19.
- The developed solution proved to be efficient in reading values, capable of demonstrating information of: number of patients, number of readings, measurement date and time, monitoring by device, evolution in the last 7 days, cities with the highest index (for state or national analyzes), patient age, oximetry, BPM and patient concentration heat maps. The solution proves to be affordable in economic terms.

3.2 Block diagram



HARDWARE TOOLS:

- Nodemcu
- Spo2 Sensor
- Heartbeat sensor
- Pressure sensor
- Servo motor
- Buzzer

Software required:

- Arduino ide
- Embedded C
- Ubidots cloud

CHAPTER 4

EXPERIMENTAL SETUP AND IMPLEMENTATION OF PROJECT

Description:

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. Arduino code is written in Embedded C with an addition of special methods and functions, which we'll mention later on. Embedded C is a human-readable programming language. When you create a 'sketch' (the name given to Arduino code files), it is processed and compiled to machine language. Arduino IDE(Integrated Development Environment) is the software for Arduino. ... It is used for writing code, compiling the code to check if any errors are there and uploading the code to the Arduino. It is a cross-platform software which is available for every Operating System like Windows, Linux, macOS.

Ubidots is an Internet of Things (IoT) Development Platform that empowers engineers and System Integrators to build IoT applications that turn sensor data into actionable insights. Our technology stack puts together backend and frontend ingredients to accelerate IoT innovation. Equipped with device friendly APIs (accessed over HTTP, MQTT, TCP or UDP protocols), Ubidots provides a simple and secure connection for sending and retrieving data to and from our cloud service in real-time. Ubidots gives decision makers the exact information needed for critical decisions in real-time, regardless of where the users are. ... Today, data at your fingertips is the norm and Ubidots agrees. Access your applications from any Android device and stay informed when on the go.

4.1. EMBEDDED SYSTEMS

4.1.1 Overview of embedded systems

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems have become very important today as they control many of the common devices we use.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

Embedded systems provide several functions

- Monitor the environment; embedded systems read data from input sensors. This data is then processed and the results displayed in some format to a user or users
- Control the environment; embedded systems generate and transmit commands for actuators.

- Transform the information; embedded systems transform the data collected in some meaningful way, such as data compression/decompression

Although interaction with the external world via sensors and actuators is an important aspect of embedded systems, these systems also provide functionality specific to their applications. Embedded systems typically execute applications such as control laws, finite state machines, and signal processing algorithms. These systems must also detect and react to faults in both the internal computing environment as well as the surrounding electromechanical systems.

There are many categories of embedded systems, from communication devices to home appliances to control systems. Examples include;

- Communication devices
 - e. g.: modems, cellular phones
- Home Appliances
 - e. g.: CD player, VCR, microwave oven
- Control Systems
 - e. g.: Automobile anti-lock braking systems, robotics, satellite control

4.1.2 Block diagram of an embedded system:

An embedded system usually contains an embedded processor. Many appliances that have a digital interface -- microwaves, VCRs, cars -- utilize embedded systems. Some embedded systems include an operating system. Others are very specialized resulting in the entire logic being implemented as a single program. These systems are embedded into some device for some specific purpose other than to provide general purpose computing . A typical embedded system is shown in Fig 3.1.2

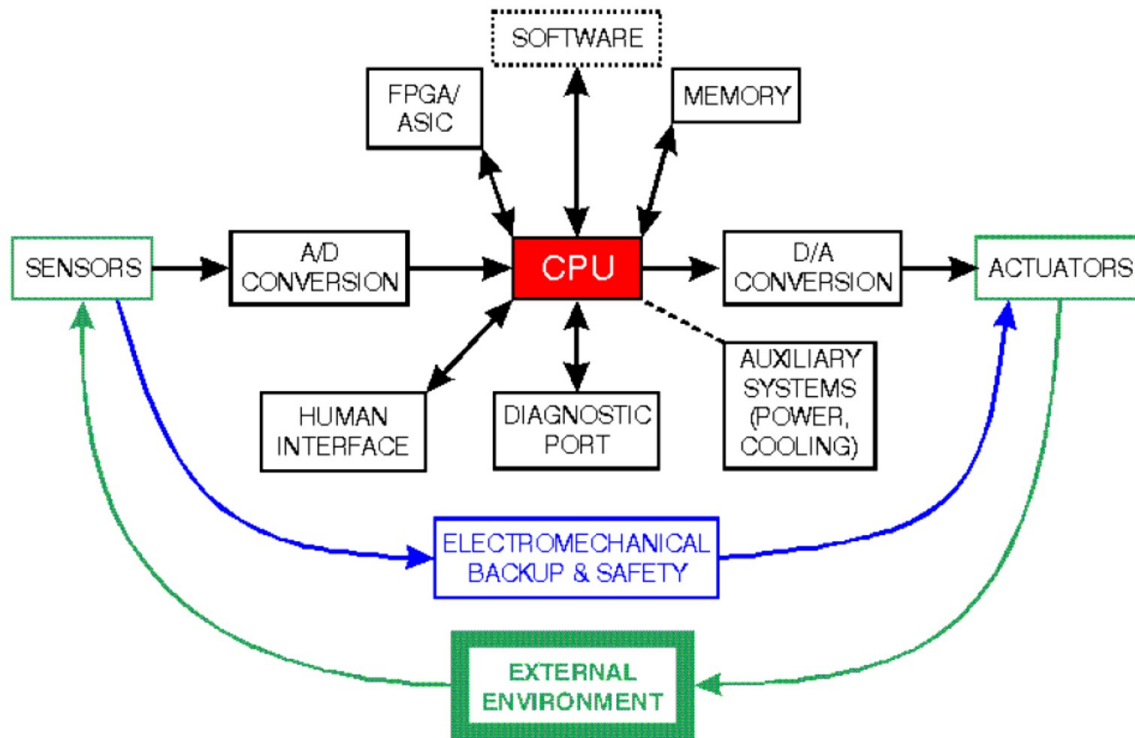


Fig 4.1.2 Block diagram of a typical embedded system

4.1.3 Characteristics of embedded systems

Embedded systems are characterized by a unique set of characteristics. Each of these characteristics imposed a specific set of design constraints on embedded systems designers. The challenge to designing embedded systems is to conform to the specific set of constraints for the application.

Application Specific Systems:

Embedded systems are not general-purpose computers. Embedded system designs are optimized for a specific application. Many of the job characteristics are known before the hardware is designed. This allows the designer to focus on the specific design constraints of a well-defined application. As such, there is limited user reprogram ability. Some embedded systems, however, require the flexibility of reprogram ability. Programmable DSPs are common for such applications.

Reactive Systems

As mentioned earlier, a typical embedded systems model responds to the environment via sensors and control the environment using actuators. This requires

embedded systems to run at the speed of the environment. This characteristic of embedded system is called “reactive”. Reactive computation means that the system (primarily the software component) executes in response to external events. External events can be either periodic or aperiodic. Periodic events make it easier to schedule processing to guarantee performance. Aperiodic events are harder to schedule. The maximum event arrival rate must be estimated in order to accommodate worst case situations. Most embedded systems have a significant reactive component. One of the biggest challenges for embedded system designers is performing an accurate worst case design analysis on systems with statistical performance characteristics (e.g., cache memory on a DSP or other embedded processor). Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. Systems with this requirement must often design to worst case performance. But accurately predicting the worst case may be difficult on complicated architectures. This often leads to overly pessimistic estimates erring on the side of caution. Many embedded systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements. Many real-time systems are also reactive systems.

Distributed Systems

A common characteristic of an embedded system is one that consists of communicating processes executing on several CPUs or ASICs which are connected by communication links. The reason for this is economy. Economical 4 8-bit microcontrollers may be cheaper than a 32-bit processors. Even after adding the cost of the communication links, this approach may be preferable. In this approach, multiple processors are usually required to handle multiple time-critical tasks. Devices under control of embedded systems may also be physically distributed.

4.1.4 Heterogeneous Architectures

Embedded systems often are composed of heterogeneous architectures (Fig 1.2). They may contain different processors in the same system solution. They may also be mixed signal systems. The combination of I/O interfaces, local and remote memories, and sensors and actuators makes embedded system design truly unique. Embedded systems also have tight design constraints, and heterogeneity provides better design flexibility.

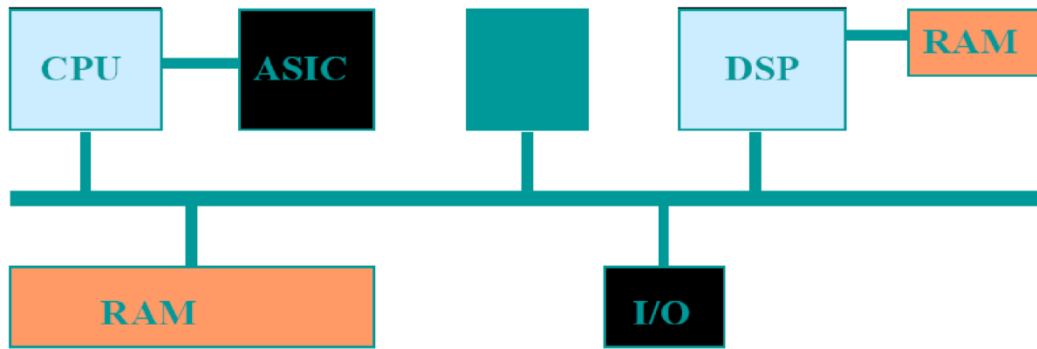


Fig 4.1.4 Embedded Systems having Heterogeneous Architectures

Harsh environment

Many embedded systems do not operate in a controlled environment. Excessive heat is often a problem, especially in applications involving combustion (e.g., many transportation applications). Additional problems can be caused for embedded computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse.

System safety and reliability

As embedded system complexity and computing power continue to grow, they are starting to control more and more of the safety aspects of the overall system. These safety measures may be in the form of software as well as hardware control. Mechanical safety backups are normally activated when the computer system loses control in order to safely shut down system operation. Software safety and reliability is a bigger issue. Software doesn't normally "break" in the sense of hardware. However software may be so complex that a set of unexpected circumstances can cause software failures leading to unsafe situations. Discussion of this topic is outside the scope of this book, but the challenges for embedded designers include designing reliable software and building cheap, available systems using unreliable components. The main challenge for embedded system designers is to obtain low-cost reliability with minimal redundancy.

Control of physical systems

One of the main reasons for embedding a computer is to interact with the environment. This is often done by monitoring and controlling external machinery. Embedded computers transform the analog signals from sensors into digital form for

processing. Outputs must be transformed back to analog signal levels. When controlling physical equipment, large current loads may need to be switched in order to operate motors and other actuators. To meet these needs, embedded systems may need large computer circuit boards with many non-digital components. Embedded system designers must carefully balance system trade offs among analog components, power, mechanical, network, and digital hardware with corresponding software.

4.1.5 Small and low weight

Many embedded computers are physically located within some larger system. The form factor for the embedded system may be dictated by aesthetics. For example, the form factor for a missile may have to fit inside the nose of the missile. One of the challenges for embedded systems designers is to develop non-rectangular geometries for certain solutions. Weight can also be a critical constraint. Embedded automobile control systems, for example, must be light weight for fuel economy. Portable CD players must be light weight for portability purposes.

Cost sensitivity

Cost is an issue in most systems, but the sensitivity to cost changes can vary dramatically in embedded systems. This is mainly due to the effect of computer costs have on profitability and is more a function of the proportion of cost changes compared to the total system cost.

Power management

Embedded systems have strict constraints on power. Given the portability requirements of many embedded systems, the need to conserve power is important to maintain battery life as long as possible. Minimization of heat production is another obvious concern for embedded systems.

4.2 UBIDOTS EXPLORER:

4.2.1 Introduction

UBIDOTS is an IoT Platform empowering innovators and industries to prototype and scale IoT projects to production. Use the UBIDOTS platform to send data to the cloud from any Internet-enabled device. You can then configure actions and alerts based on your real-time data and unlock the value of your data through visual tools. UBIDOTS offers a REST API that allows you to read and write data to the resources available: data sources, variables, values, events and insights. The API supports both HTTP and HTTPS and an API Key is required.

Your data will be protected with two more replication, encrypted storage and optional TLS/SSL data support. You can also customize permission groups to each module of the platform, making sure the right information is shown to the right user.

Every time a device updates a sensor value in a variable, a data-point or "**dot**" is created. UBIDOTS stores dots that come from your devices inside variables, and these stored dots have corresponding **timestamps**:

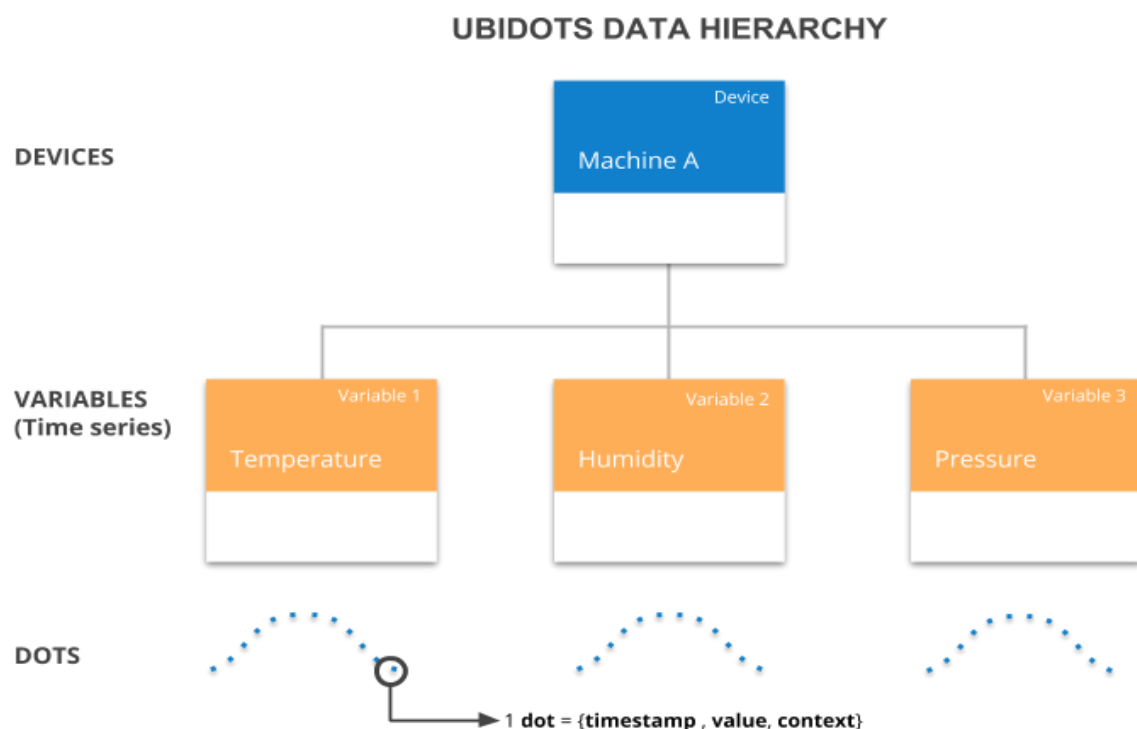


FIG 4.2.1 DATA HIERARCHY

Each dot contains these items:

Item	Description	Mandatory
value	A numerical value. Ubidots accepts up to 16 floating-point length numbers.	YES
timestamp	Unix Epoch time, in milliseconds. If not specified, then our servers will assign one upon reception.	NO
context	An arbitrary collection of key-value pairs. Mostly used to store the latitude and longitude coordinates of GPS devices.	NO

Values

A numerical value. Ubidots accepts up to 16 floating-point length numbers.

```
{"value" : 34.87654974}
```

Timestamps

A **timestamp**, as best described here, is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. Please keep in mind that when you send data to Ubidots, you must set the timestamp in milliseconds; also, if you retrieve a dot's timestamp, it will be in milliseconds.

```
"timestamp" : 1537453824000
```

The above timestamp corresponds to Thursday, September 20, 2018 2:30:24 PM.

PRO-TIP: A useful tool to convert between Unix timestamps and human-readable dates is **Epoch Converter**.

Context

Numerical values are not the only data type supported; you can also store *string* or *char* data types inside what we call **context**. The **context** is a key-

value object that allows you to store not only numerical but also string values. An example use of the **context** could be:

```
"context" : {"status" : "on", "weather" : "sunny"}
```

A **context** is commonly used to store the latitude and longitude coordinates of your device for GPS/tracking application use cases. All Ubidots maps uses the *lat* and *lng* keys from a dot's **context** to extract the coordinates of your device, in that way you just need to send a single dot with the coordinates values in the variable context to plot a map instead of sending separately both latitude and longitude in two different variables. Below you can find a typical **context** with coordinates values:

```
"context" : {"lat":-6.2, "lng":75.4, "weather" : "sunny"}
```

Please note that you can mix both string and numerical values in the **context**. If your application is for geo-localization purposes, make sure that the coordinates are set in decimal degrees.

Time Series

Based on the above, we can illustrate an Ubidots time series like this:

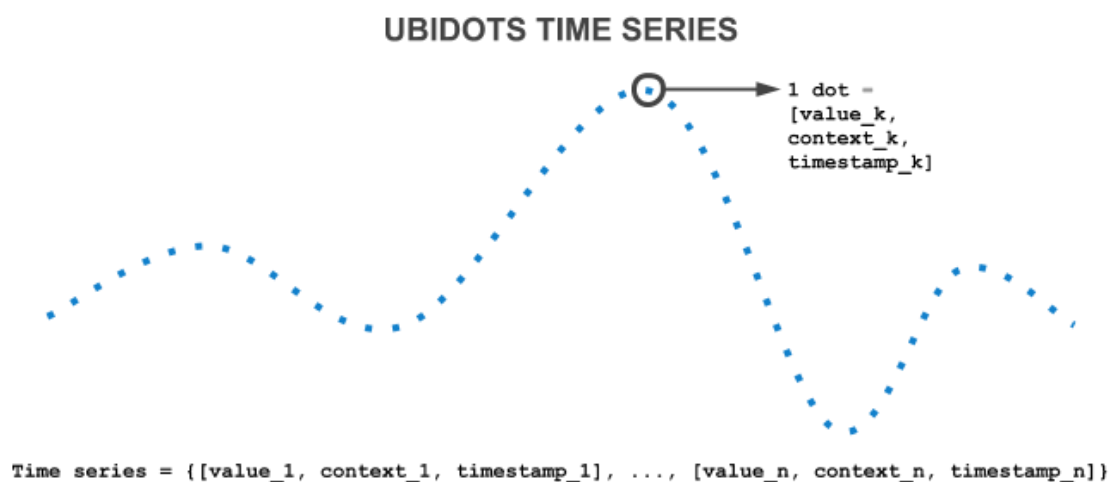


FIG 4.21 UBIDOTS TIME SERIES

The basics components of any Internet of Things application powered by Ubidots are: **Devices**, **Variables**, **Synthetic Variables Engine**, **Dashboards**, and **Events**. Within this article we will address each of these concepts as they relate to Ubidots

IoT Development and Deployment Platform and how you can better organize your Ubidots Apps to best connect with the users.

Once your devices, variables, and general Application is assembled, give your App some layers with Ubidots Device Management to learn more about Ubidots internal architecture and how you can use Apps, Organizations, and Users to efficiently connect your data with those who should be using it.

1. Devices
2. Variables
3. Synthetic Variables Engine
4. Dashboard and Visualizations
5. Events.

4.2.1. Devices

A Ubidots' device is a virtual representation of a data-source or simply, an asset taking sensor data and transmitting said data through a connection protocol to Ubidots' cloud. Click [here](#) for current firmware examples and tutorials for connecting your device to Ubidots.

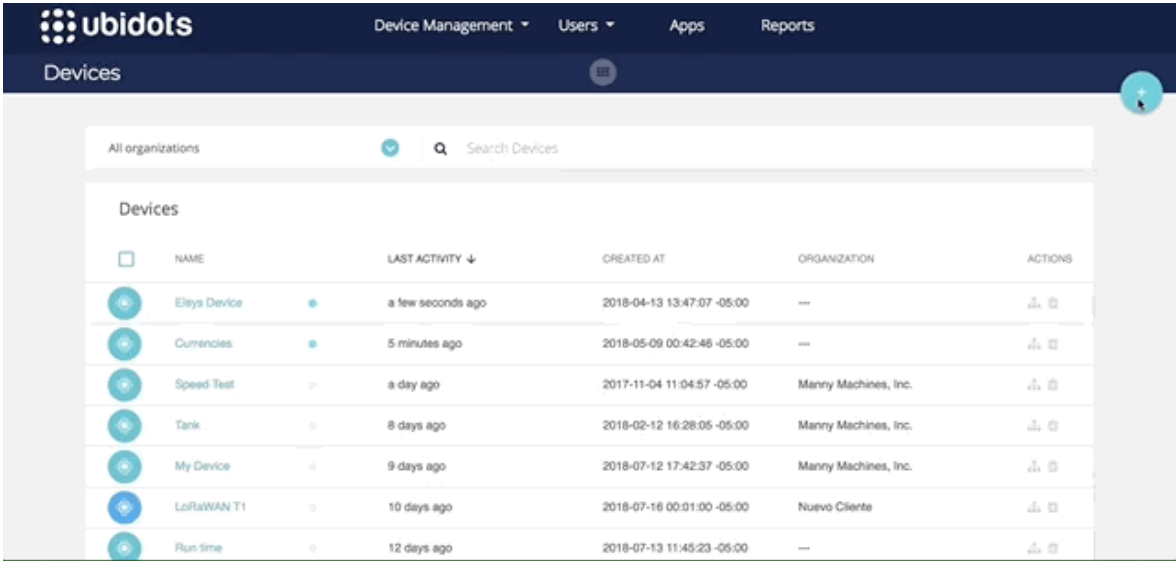
All devices are different, but the standard setup on any device entails:

A library that should be installed in the device's IDE. Filling parameters for authentication and connection such as a TOKEN (i.e. the unique ID for each account or user), device and variable labels (i.e. devices and variables unique identifiers in Ubidots), Wi-Fi SSID and Password, depending on the device and the requirements. Making API request (i.e. a call from a device to the web server). Ubidots' libraries spare the need to manually make these requests. However, if you're working with a device not currently in the list of supported hardware, you can choose a connectivity protocol such as MQTT or HTTP and make a request using the API accordingly.

Creating devices: there are three ways to create devices:

1. Devices are created automatically in Ubidots the first time a dot is received to a user's private TOKEN or an Organizational TOKEN.
2. Clicking the "+" icon in the top right corner of any single device's screen.

3. Creating a Device Type which then creates a new device with pre-determined device properties, appearance, and variables and automate the on-boarding of thousands of devices.



The screenshot shows the Ubidots web interface. At the top is a dark blue header with the Ubidots logo and navigation links: Device Management, Users, Apps, and Reports. Below the header is a 'Devices' section with a search bar and a table of devices. The table has columns for NAME, LAST ACTIVITY, CREATED AT, ORGANIZATION, and ACTIONS. The devices listed are: Ellys Device, Currencies, Speed Test, Tank, My Device, LoRaWAN T1, and Run time.

NAME	LAST ACTIVITY	CREATED AT	ORGANIZATION	ACTIONS
Ellys Device	a few seconds ago	2018-04-13 13:47:07 -05:00	---	[Icons]
Currencies	5 minutes ago	2018-05-09 00:42:46 -05:00	---	[Icons]
Speed Test	a day ago	2017-11-04 11:04:57 -05:00	Manny Machines, Inc.	[Icons]
Tank	8 days ago	2018-02-12 16:28:05 -05:00	Manny Machines, Inc.	[Icons]
My Device	9 days ago	2018-07-12 17:42:37 -05:00	Manny Machines, Inc.	[Icons]
LoRaWAN T1	10 days ago	2018-07-16 00:01:00 -05:00	Nuevo Cliente	[Icons]
Run time	12 days ago	2018-07-13 11:45:23 -05:00	---	[Icons]

FIG 4.22 CREATING DEVICES

4.2.2. Variables

Once a device is created and receiving data from your hardware or another 3rd party data-source, the data will be presented in its raw or calculated form as a **variable**.

Types of Variables:

- **Default** - raw data coming from devices (people counted).
- **Synthetic** - correspond to statistical or arithmetical operations of default variables in a determined time-frame (e.g. average daily traffic this month).

Below you can read more about synthetic variable setups.

Creating variables: there are two ways to create variables.

1. Assigning a label as a variable within your hardware's code.
2. Clicking the "+" icon found in any single device screen and assign a name which will also correspond to the variable's label. By default, Ubidots assigns an additional unique identifier to the variable called Variable ID. To

find both the variable's ID and variable's label, select the "i" icon form the variables card.

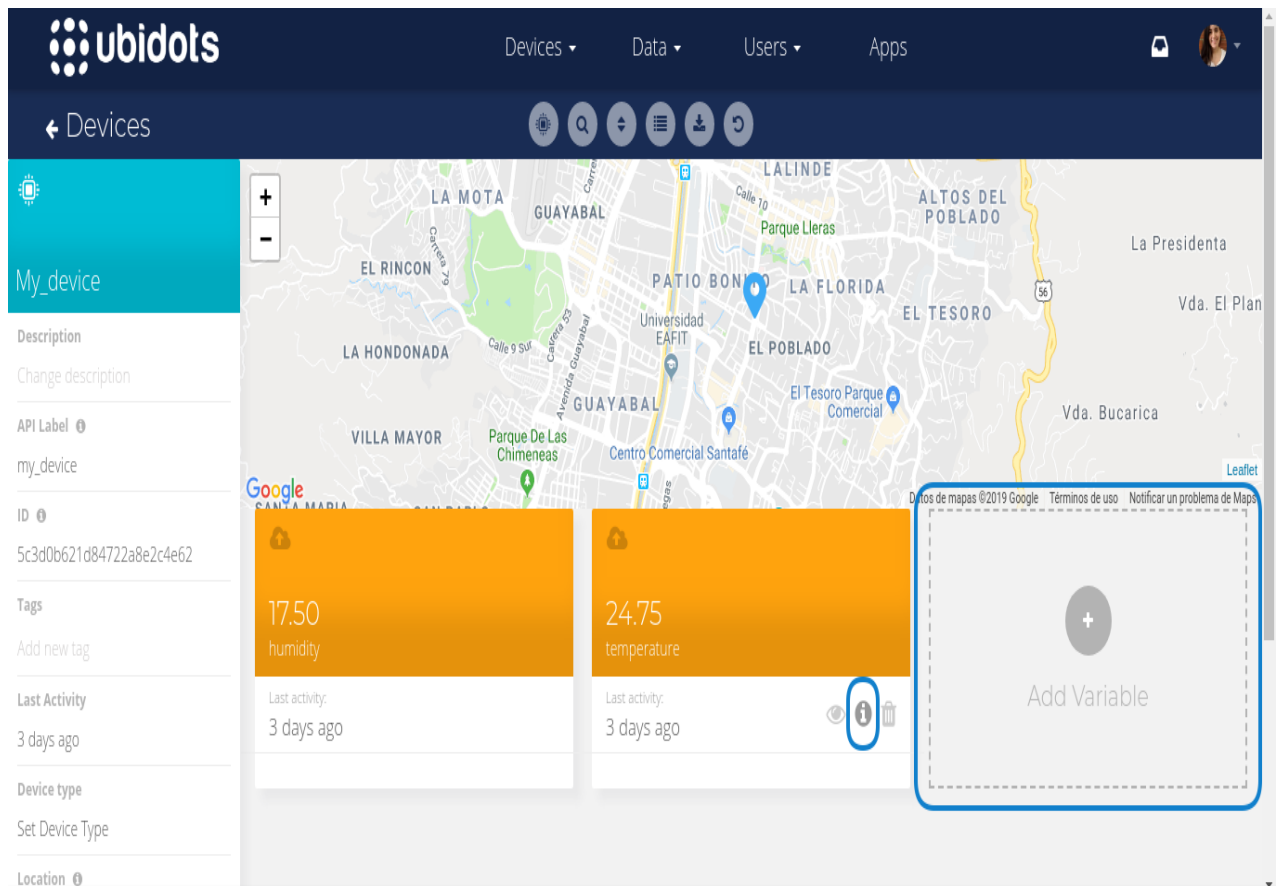


FIG 4.2.2 ADDING VARIABLES

4.2.3. Synthetic Variables Engine

Ubidots Synthetic Variables (SV) Engine is a simple and effective way to create insights from the raw data stored in Ubidots' Cloud. Using Ubidots SV Engine, users can create **Synthetic Variables** that calculate and enhance data to give data-driven decision to user of an Ubidots powered application.

For example, to calculate Average (mean) Energy consumed this week, we will need to create a synthetic variable called Energy to calculate the product of Current and Voltage ($P=C*V$): the **default variable Current is multiplied by Voltage**. Then average the product every week with built-in function *mean()*.

Synthetic Variables Analytics

Energy

Synthetic Variable

Enter your synthetic variable expression. [Learn more about synthetic variables.](#)

1 mean(

current

*

voltage

, "W")

Timezone:

America/Bogota

Click on a variable to add it to your expression:

Current

Voltage

Cancel

Accept

FIG 4.2.3 SYNTHETIC VARIABLE

Here is a complete list of available functions, such as **cos(x)** or **ceil(x)** functions, and more!

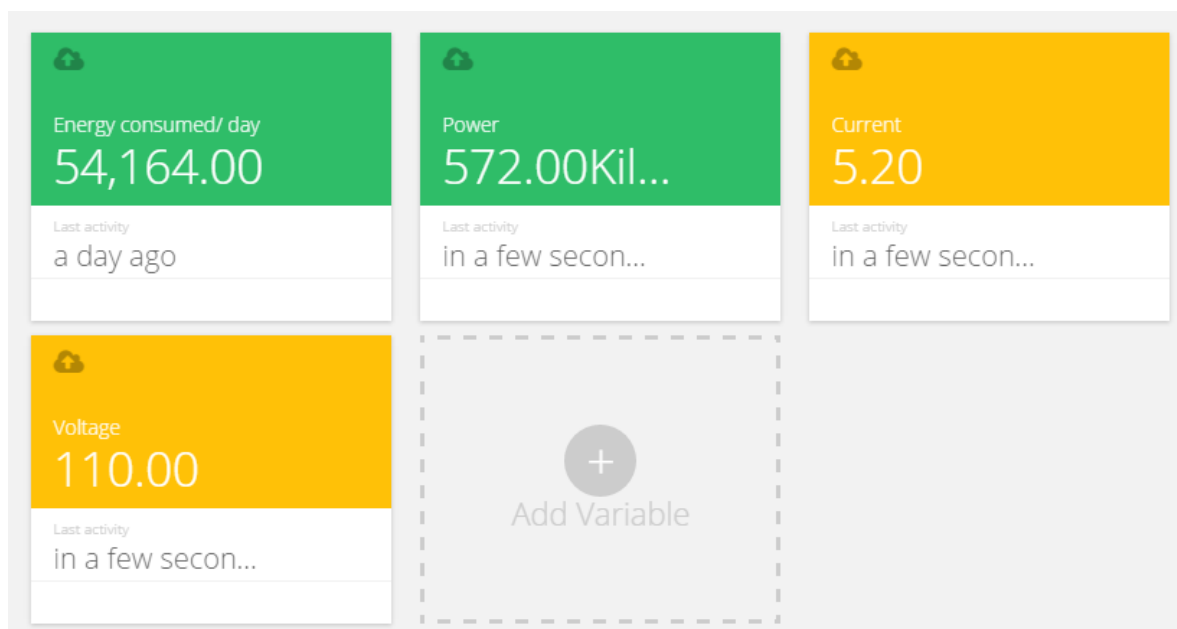


FIG 4.2.4 DIFF BTW DEFAULT AND SYNTHETIC VARIABLES

Note that default variables are yellow, and Synthetic Variables are green.

4.2.4. Dashboards and Visualizations

Dashboards are the human-machine interfaces where data is easily visualized. Your Ubidots account will let you create as many dashboards as needed, containing widgets and data-visualizations to comprehend your data at a glance. For additional details on your dashboard's construction, check out this tutorial.

4.2.5. Events

In Ubidots, Events are messages triggered and delivered through Email, SMS, Telegrams, Slack, Voice Call or webhook messages based on a customized design rule created in the application.

3 Event types:

- When sensor readings reach a specific value:

The screenshot shows the 'If triggers' configuration bar. The first dropdown is set to 'demo'. The second dropdown is 'Value'. The third dropdown is 'is', which is open, showing a list of comparison operators: 'Equal to' (selected), 'Equal to', 'Greater than', 'Greater than or equal to', 'Less than', 'Less than or equal to', and 'Not equal to'. The fourth input field is '10', followed by 'for' and the fifth input field '0', and finally 'minutes'. A trash icon is on the right.

- Variable is inactive during a specific amount of time:

The screenshot shows the 'If triggers' configuration bar. The first dropdown is 'demo'. The second dropdown is 'Has been inactive'. The third dropdown is 'for', followed by the input field '10', and finally 'minutes'. Below the configuration bar are two buttons: '+ And' and '+ Or'. A trash icon is on the right.

- When a device enters or exits a specific area (geofence):

FIG 4.2.5 EVENTS

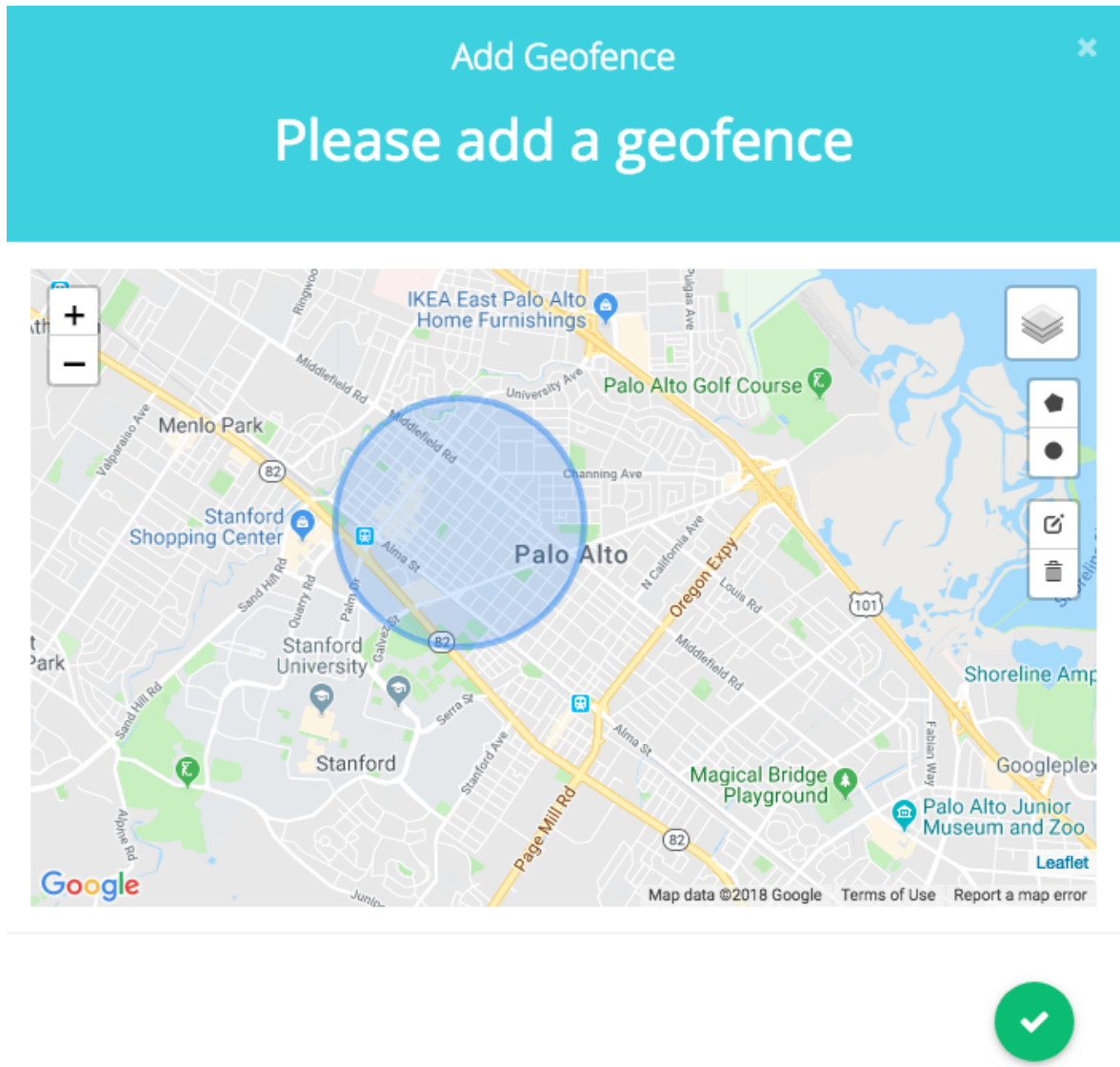


FIG 4.2.6 GEOFENCE

You can also determine the format of alerts by choosing between emails, SMS, Telegrams or Webhook.

HARDWARE COMPONENTS:

4.3 NODEMCU

NODEMCU is an open source IoT platform. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The term "NODEMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS

NodeMCU was created shortly after the ESP8266 came out. On December 30, 2013, Espressif Systems began production of the ESP8266. The ESP8266 is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications (see related projects). NodeMCU started on 13 Oct 2014, when Hong committed the first file of nodemcu-firmware to GitHub. Two months later, the project expanded to include an open-hardware platform when developer Huang R committed the gerber file of an ESP8266 board, named devkit v0.9. Later that month, Tuan PM ported MQTT client library from Contiki to the ESP8266 SoC platform, and committed to NodeMCU project, then NodeMCU was able to support the MQTT IoT protocol, using Lua to access the MQTT broker. Another important update was made on 30 Jan 2015, when Devsaurus ported the u8glib to NodeMCU project, enabling NodeMCU to easily drive LCD, Screen, OLED, even VGA displays.

In summer 2015 the creators abandoned the firmware project and a group of independent contributors took over. By summer 2016 the NodeMCU included more than 40 different modules. Due to resource constraints users need to select the modules relevant for their project and build a firmware tailored to their needs.

ESP8266 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 toolchains 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 ESP8266 enthusiasts developed an Arduino core for the ESP8266 WiFi SoC, popularly called the "ESP8266 Core for the Arduino IDE". This has become a leading software development platform for the various ESP8266-based modules and development boards, including NODUMCUs.

4.3.1 INTRODUCTION

NODEMCU is an open source LUA based firmware developed for ESP8266 wifi chip. By exploring functionality with ESP8266 chip, NODEMCU firmware comes with ESP8266 Development board/kit i.e. NODEMCU Development board.

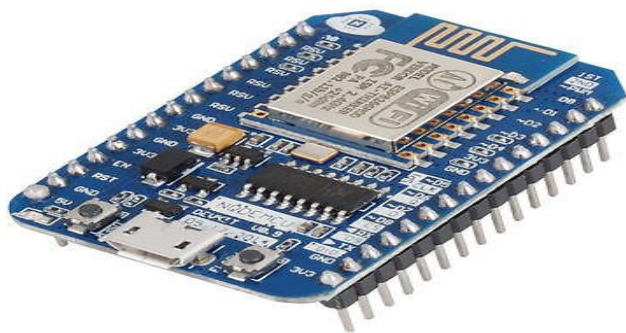


Fig 4.3.1 NODEMCU IMG

4.3.2 NODEMCU Development Board/kit v0.9 (Version1)

Since NODEMCU is open source platform, their hardware design is open for edit/modify/build.

NODEMCU Dev Kit/board consist of ESP8266 wifi enabled chip. The **ESP8266** is a low-cost Wi-Fi chip developed by Espressif Systems with TCP/IP protocol. For more information about ESP8266, you can refer ESP8266 WiFi Module.

There is Version2 (V2) available for NodeMCU Dev Kit i.e. **NodeMCU Development Board v1.0 (Version2)**, which usually comes in black colored PCB.



FIG 4.3.2 BACK VIEW OF NODEMCU

NodeMCU Development Board/kit v1.0 (Version2)

For more information about NodeMCU Boards available in market refer NodeMCU Development Boards NodeMCU Dev Kit has **Arduino like** Analog (i.e. A0) and Digital (D0-D8) pins on its board.

It supports serial communication protocols i.e. UART, SPI, I2C etc.

Using such serial protocols we can connect it with serial devices like I2C enabled LCD display, Magnetometer HMC5883, MPU-6050 Gyro meter + Accelerometer, RTC chips, GPS modules, touch screen displays, SD cards etc.

How to start with NodeMCU?

NodeMCU Development board is featured with wifi capability, analog pin, digital pins and serial communication protocols.

To get start with using NodeMCU for IoT applications first we need to know about how to write/download NodeMCU firmware in NodeMCU Development Boards. And before that where this NodeMCU firmware will get as per our requirement.

There is online NodeMCU custom builds available using which we can easily get our custom NodeMCU firmware as per our requirement.

To know more about how to build custom NodeMCU firmware online and download it refer [Getting started with NodeMCU](#)

How to write codes for NodeMCU?

After setting up ESP8266 with Node-MCU firmware, let's see the IDE (Integrated Development Environment) required for development of NodeMCU.

NodeMCU with ESPlorer IDE

Lua scripts are generally used to code the NodeMCU. Lua is an open source, lightweight, embeddable scripting language built on top of C programming language.

For more information about how to write Lua script for NodeMCU refer [Getting started with NodeMCU using ESPlorerIDE](#)

4.3.3 NodeMCU with Arduino IDE

Here is another way of developing NodeMCU with a well-known IDE i.e. Arduino IDE. We can also develop applications on NodeMCU using Arduino development environment. This makes easy for Arduino developers than learning new language and IDE for NodeMCU.

For more information about how to write Arduino sketch for NodeMCU refer [Getting started with NodeMCU using ArduinoIDE](#).

Difference in using ESPlorer and Arduino IDE

Well, there is a programming language difference we can say while developing application for NodeMCU using ESPlorer IDE and Arduino IDE.

We need to code in C/C++ programming language if we are using Arduino IDE for developing NodeMCU applications and Lua language if we are using ESPlorer IDE.

Basically, NodeMCU is Lua Interpreter, so it can understand Lua script easily. When we write Lua scripts for NodeMCU and send/upload it to NodeMCU, then they will get executes sequentially. It will not build binary firmware file of code for NodeMCU to write. It will send Lua script as it is to NodeMCU to get execute.

In Arduino IDE when we write and compile code, ESP8266 toolchain in background creates binary firmware file of code we wrote. And when we upload it to NodeMCU

then it will flash all NodeMCU firmware with newly generated binary firmware code. In fact, it writes the complete firmware.

That's the reason why NodeMCU not accept further Lua scripts/code after it is getting flashed by Arduino IDE. After getting flashed by Arduino sketch/code it will be no more Lua interpreter and we got error if we try to upload Lua scripts. To again start with Lua script, we need to flash it with NodeMCU firmware.

Since Arduino IDE compile and upload/writes complete firmware, it takes more time than ESPlorer IDE.

4.4 PULSE SENSOR

The Pulse Sensor is a well-designed low-power plug-and-play heart-rate sensor for the Arduino. It can be used by students, artists, athletes, manufacturers, and game & mobile developers who want to incorporate live heart-rate data into their projects.

And the best part is that this sensor plugs right into Arduino and easily clips onto a fingertip or earlobe. It is also super small (button-shaped) with holes, so it can be sewn into fabric.



Fig 4.4.1 PULSE SENSOR

4.4.2 HARDWARE OVERVIEW

The front of the sensor is the side with the heart logo. This is where you place your finger. On the front side you will see a small round hole, from where the Kingbright's reverse mounted green LED shines.

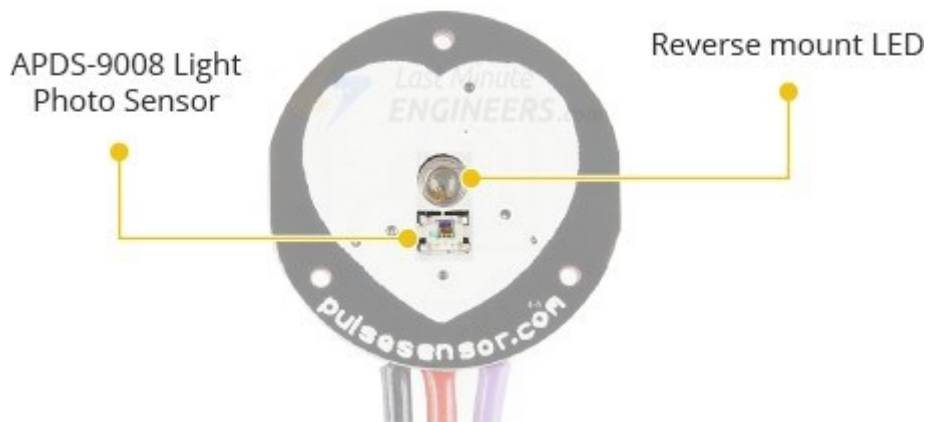


Fig 4.4.2 pulse sesor overview

Just below the LED is a small ambient light photo sensor – APDS-9008 from Avago, similar to that used in cell phones, tablets and laptops, to adjust the screen brightness in different light conditions.

On the back of the module you will find the rest of the components including a microchip's MCP6001 Op-Amp and a bunch of resistors and capacitors that make up the R/C filter network. There is also a reverse protection diode to prevent damage if the power leads are accidentally reversed.

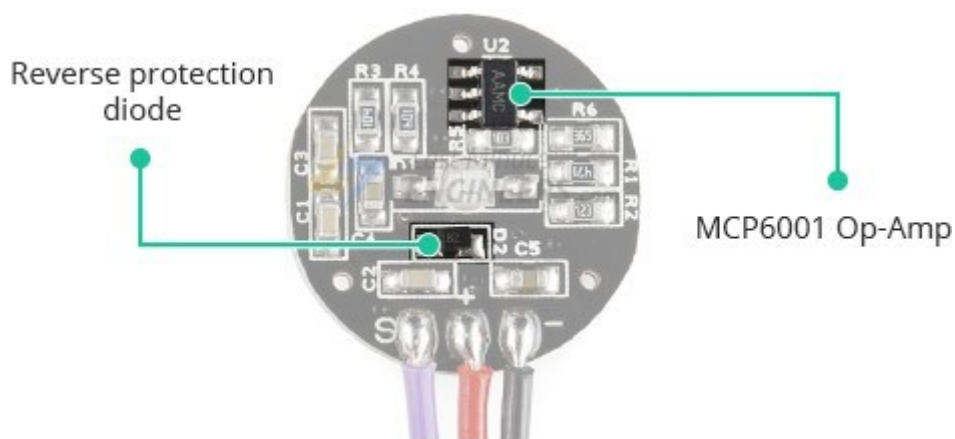


Fig 4.4.3 pulse sensor hardware

The module operates from a 3.3 to 5V DC Voltage supply with a operating current of $< 4\text{mA}$.

4.4.3 SPECIFICATION

- This is a heart beat detecting and biometric pulse rate sensor
- Its diameter is 0.625
- Its thickness is 0.125
- The operating voltage is ranges +5V otherwise +3.3V
- This is a plug and play type sensor
- The current utilization is 4mA
- Includes the circuits like Amplification & Noise cancellation
- This pulse sensor is not approved by the FDA or medical. So it is used in student-level projects, not for the commercial purpose in health issues applications.

How Pulse Sensor Works:

Optical heart-rate sensors are very easy to understand in theory. If you've ever shined a flashlight through your finger tips and seen your heartbeat pulse, you have a good handle on the theory of optical heart-rate pulse sensors.

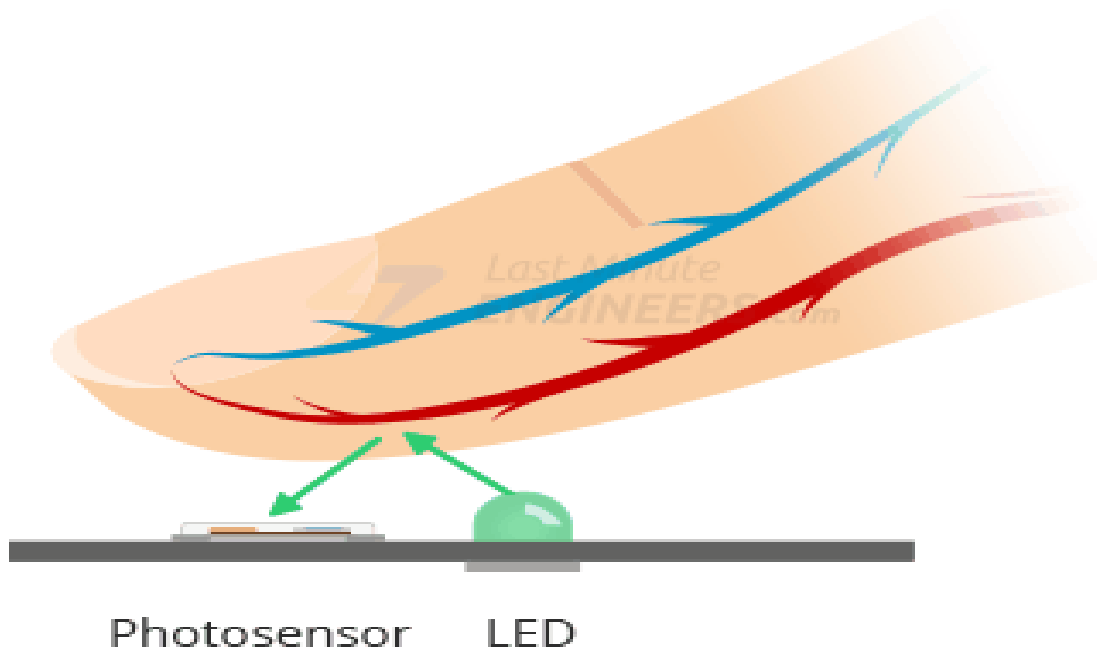


Fig 4.4.4 working of sensor

A pulse sensor or any optical heart-rate sensor, for that matter, works by shining a green light ($\sim 550\text{nm}$) on the finger and measuring the amount of reflected light using a photosensor.

This method of pulse detection through light is called Photoplethysmogram.

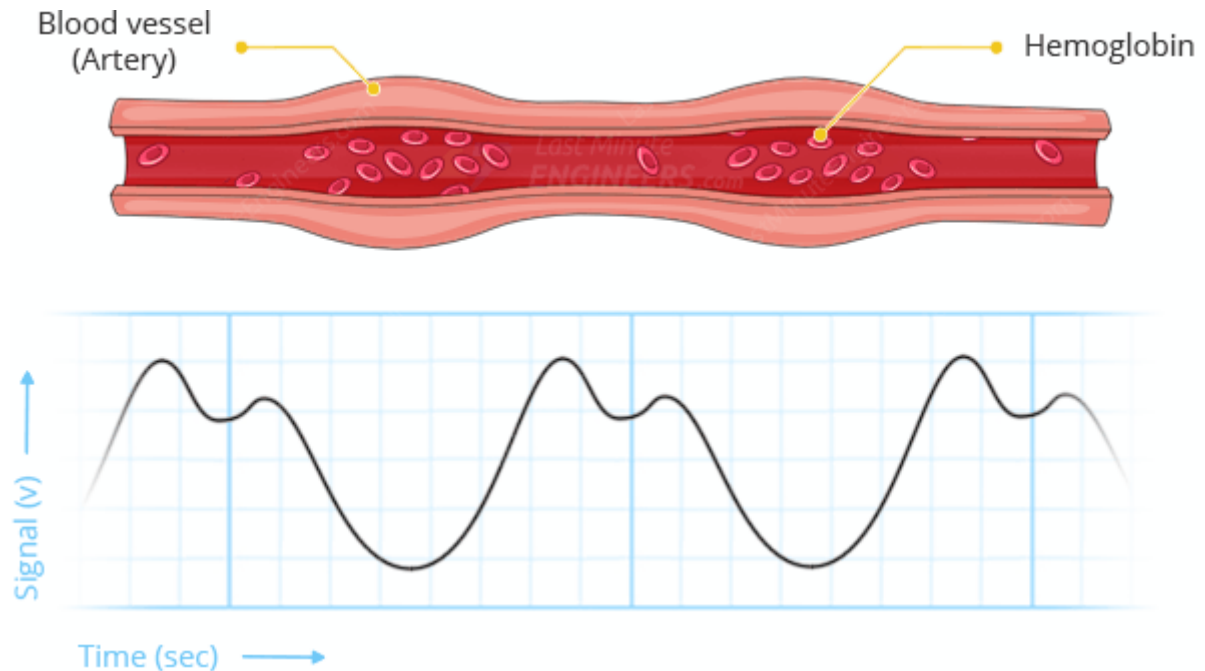


Fig 4.4.5 pulse sensor measuring readings

The oxygenated hemoglobin in the arterial blood has the characteristic of absorbing green light. The redder the blood (the higher the hemoglobin), the more green light is absorbed. As the blood is pumped through the finger with each heartbeat, the amount of reflected light changes, creating a changing waveform at the output of the photosensor.

As you continue to shine light and take photosensor readings, you quickly start to get a heart-beat pulse reading.

This signal from the photosensor is generally small and noisy, therefore the signal is passed through an R/C filter network and then amplified using an Op Amp to create a signal that is much larger, cleaner and easier to detect.

4.4.4 Pulse Sensor Pinout

The sensor comes with a 24" flat ribbon cable with 3 male header connectors. The following diagram shows the pinout.

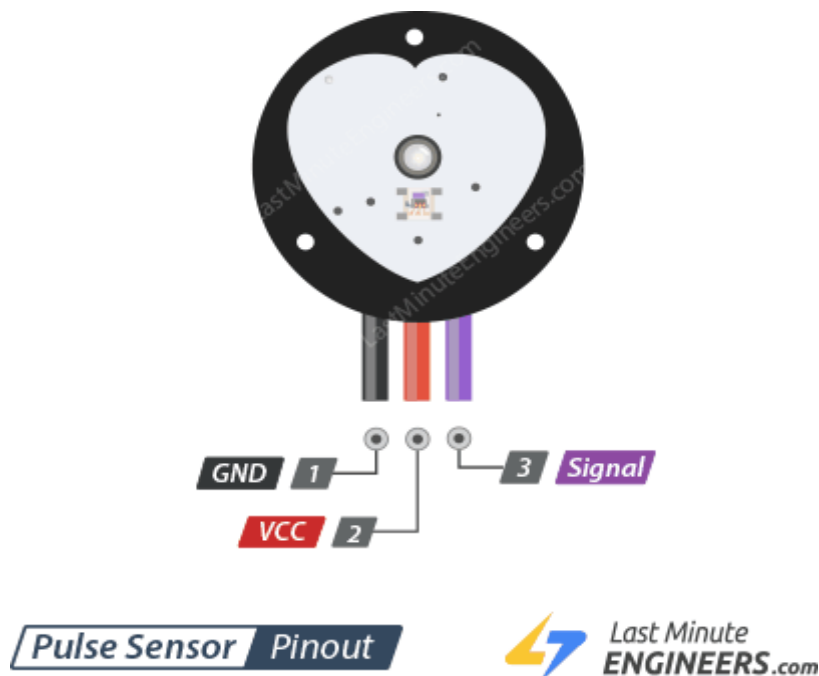


Fig 4.4.6 pulse sensor pin specification

S (Signal) is the signal output. Connects to analog input of an Arduino.

+ (VCC) is the VCC pin. Connects to 3.3 or 5V.

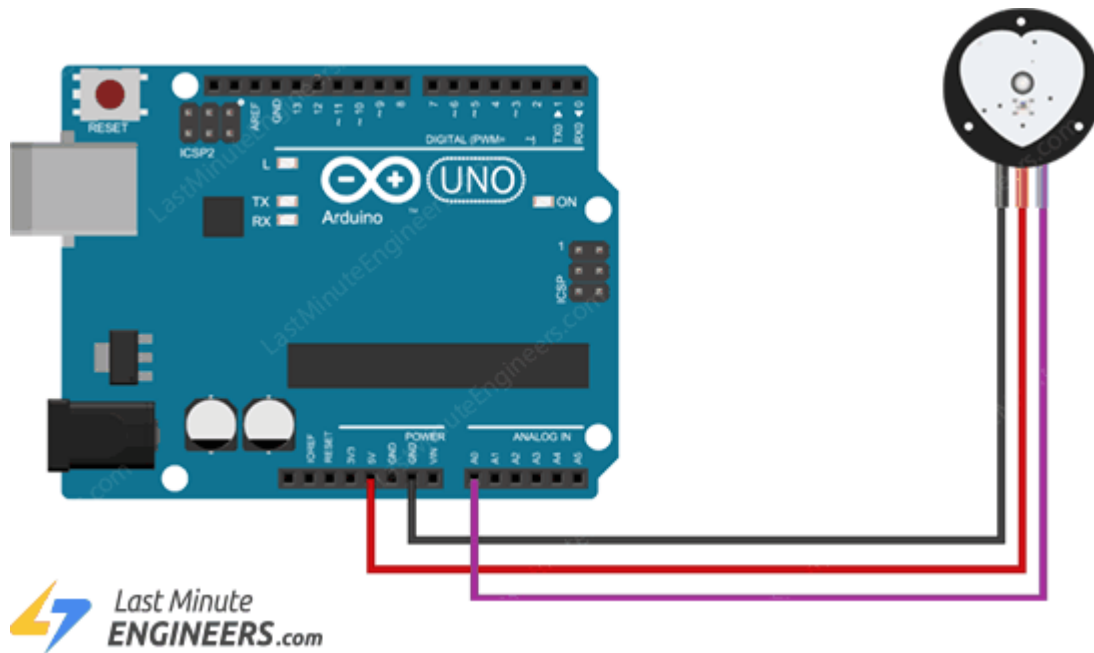
- (GND) is the Ground pin.

4.4.5 Wiring Pulse Sensor with Arduino

Hooking up the Pulse Sensor to an Arduino is super simple. You only need to connect three wires: two for power and one for reading the sensor value.

The module can be powered from 3.3 or 5V. The positive voltage connects to '+' and ground connects to '-'. The 3rd 'S' wire is the analog signal output from the sensor and this will connect to the A0 analog input of an Arduino.

Below is the hookup for the experiments with the Pulse Sensor:



4.4.7 pulse sensor wiring

Pulse Sensor Example Sketches

The Pulse sensor library has a number of example sketches. We'll go over few of them here but you can also try the other ones. You can use these example sketches as a basis for developing your own code.

To access the example sketches, navigate to the **File > Examples > Pulse sensor Playground** You will see a selection of example sketches. Select any of them to load the sketch into your IDE. Let's begin with the **Getting Started Project**.

4.4.6 Arduino Code – Heart Beat Plotting

The previous Getting Started Project sketch is designed to work with the Arduino Serial Plotter, a cool tool that comes with the Arduino IDE for visualizing the analog signals in real-time. While the sketch is running and your Arduino board is connected to your computer USB, click on: **Tools > Serial Plotter**

The signal may take a moment to settle, but you should see something like this.

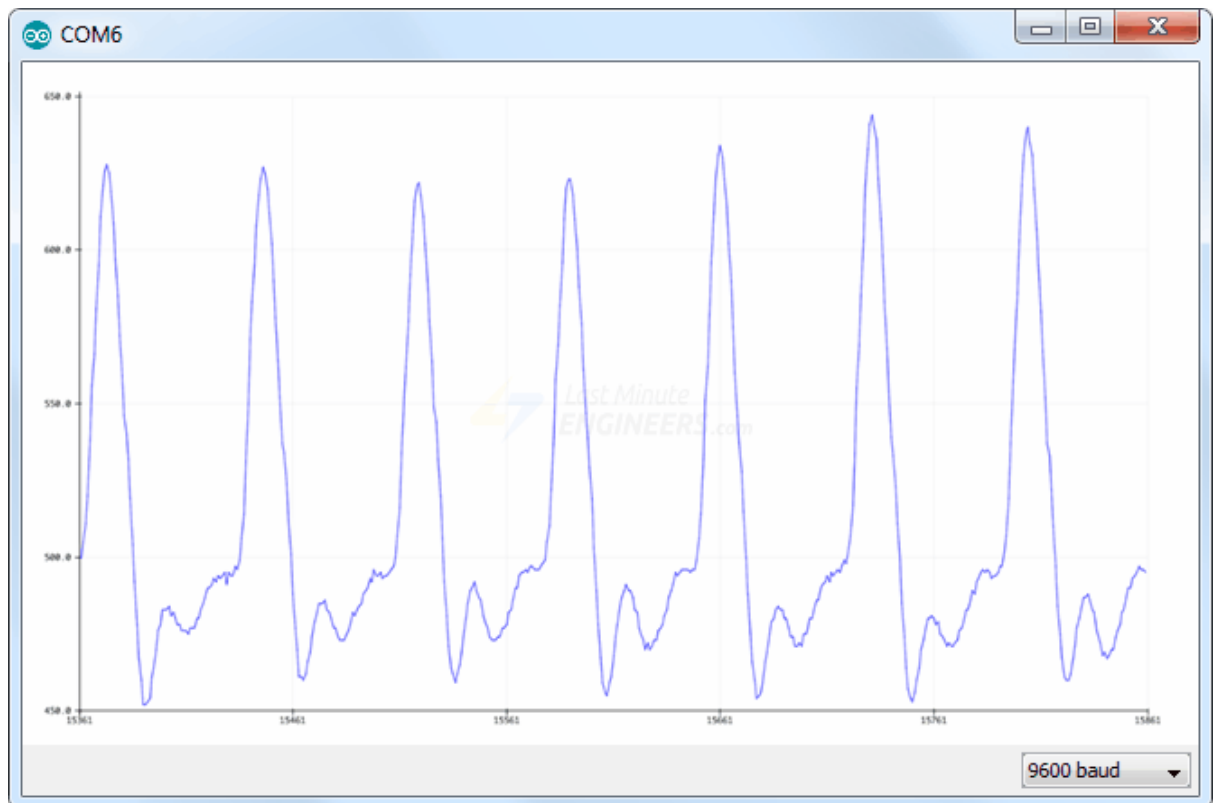


Fig 4.4.8 graphical representation of readings

Arduino Code – Measuring Heart-Rate (BPM)

For our second example, we will try to measure heart rate (Beats Per Minute or BPM). Look in the **Pulse Sensor Playground** examples menu for **Getting BPM to Monitor** and load it into your Arduino IDE.

This sketch calculates the time between pulses to obtain the heart rate and displays it on the Serial Monitor. After uploading the sketch, keep your finger as steady as possible and wait a few seconds for the readings to make sense. You will see a result like this

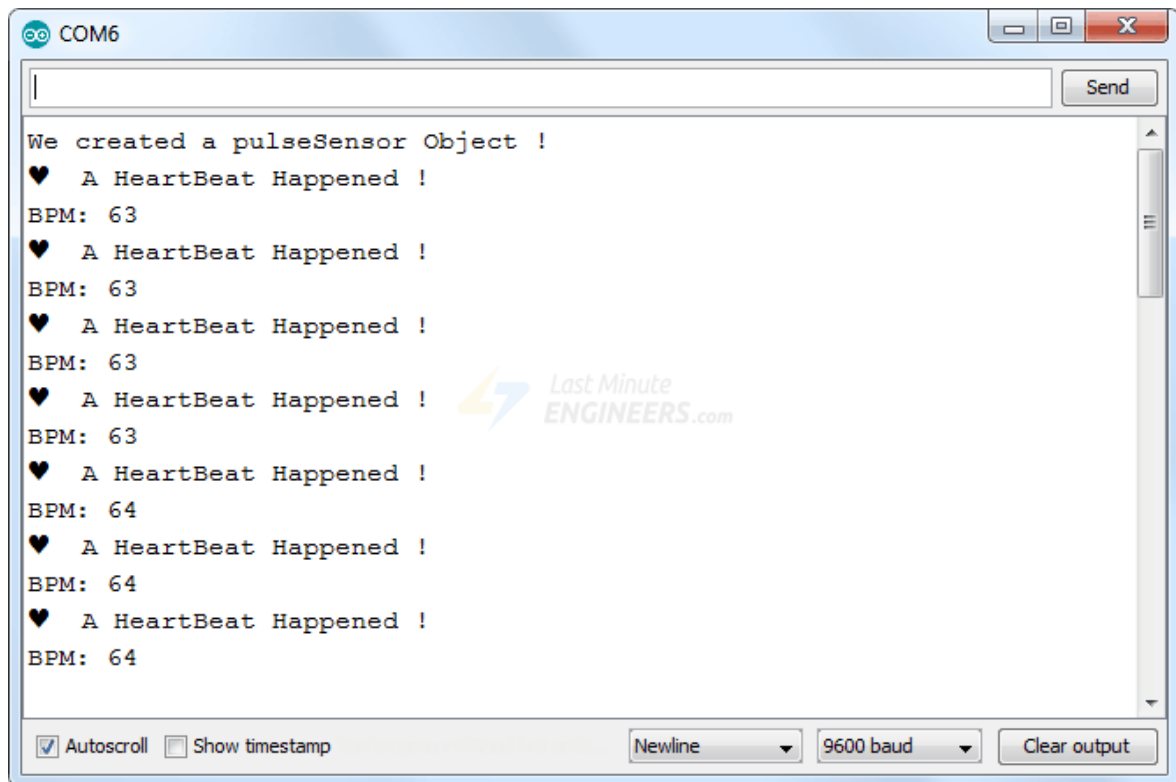


Fig 4.4.9 reading of heartbeat

4.5 MAX30100 Pulse Oximeter(spo2 sensor)



Fig 4.5.1 spo2 sensor

The sensor is **integrated pulse oximetry** and heart-rate monitor sensor solution. It combines two LED's, a photodetector, optimized optics, and low-noise analog signal processing to detect pulse and heart-rate signals. It operates from **1.8V** and **3.3V** power supplies and can be powered down through software with negligible **standby current**, permitting the power supply to remain connected at all times.

4.5.1 Working of MAX30100 Pulse Oximeter and Heart-Rate Sensor

The device has **two LEDs**, one **emitting red light**, another emitting **infrared light**. For pulse rate, only infrared light is needed. Both red light and infrared light are used to measure **oxygen levels** in the blood.

When the heart pumps blood, there is an increase in **oxygenated blood** as a result of having more blood. As the heart relaxes, the volume of oxygenated blood also decreases. By knowing the time between the increase and decrease of oxygenated blood, the **pulse rate** is determined.

It turns out, oxygenated blood absorbs more **infrared light** and passes more red light while deoxygenated blood absorbs red light and passes more infrared light. This is the main function of the MAX30100: it reads the absorption levels for both light sources and stores them in a buffer that can be read via **I2C** communication protocol.

4.5.2 Interfacing MAX30100 Pulse Oximeter Sensor with Arduino

Now let us interface MAX30100 Pulse Oximeter Sensor with Arduino and display the value in serial monitor. The circuit diagram and connection is very simple. You can follow the same.

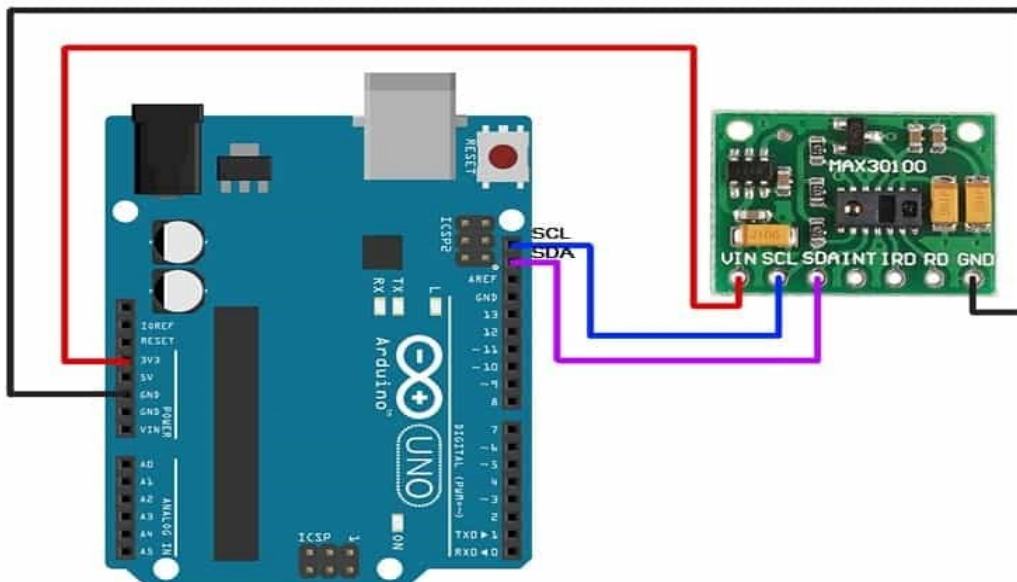


Fig 4.5.2 circuit connection

Connect the Vin pin of MAX30100 to Arduino **5V** or **3.3V** pin, GND to GND. Connect the I2C Pin of MAX30100, i.e **SCL & SDA** to **A5 & A4** of Arduino.

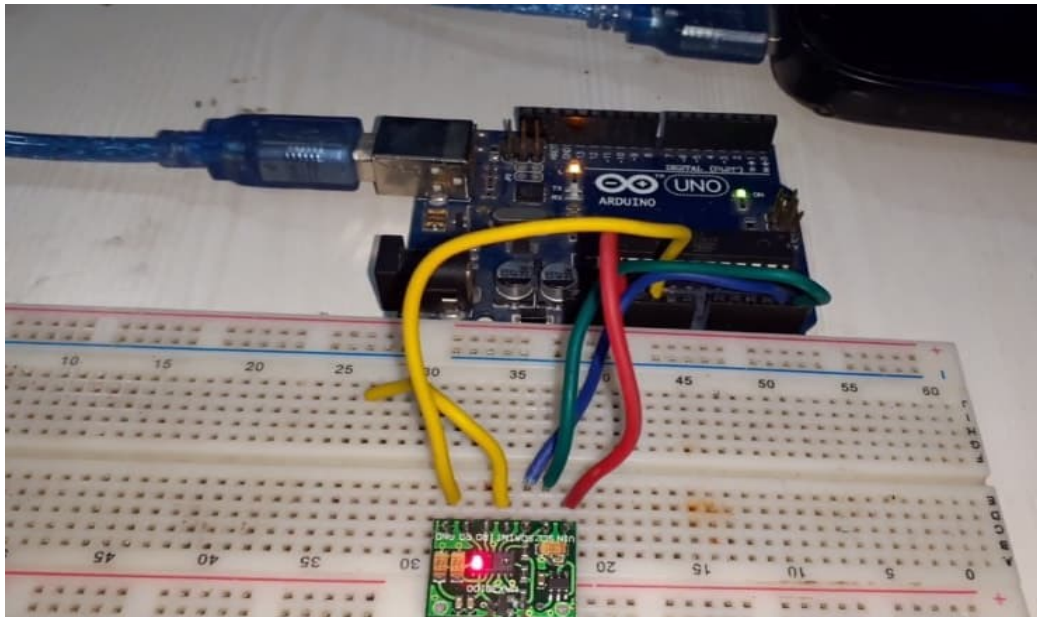


Fig 4.5.3 connecting wires on bread board

After uploading the code, open the serial monitor to see the values as shown in the image. Initially the BPM & SpO2 value appears as incorrect value but soon you can observe the correct stable reading.

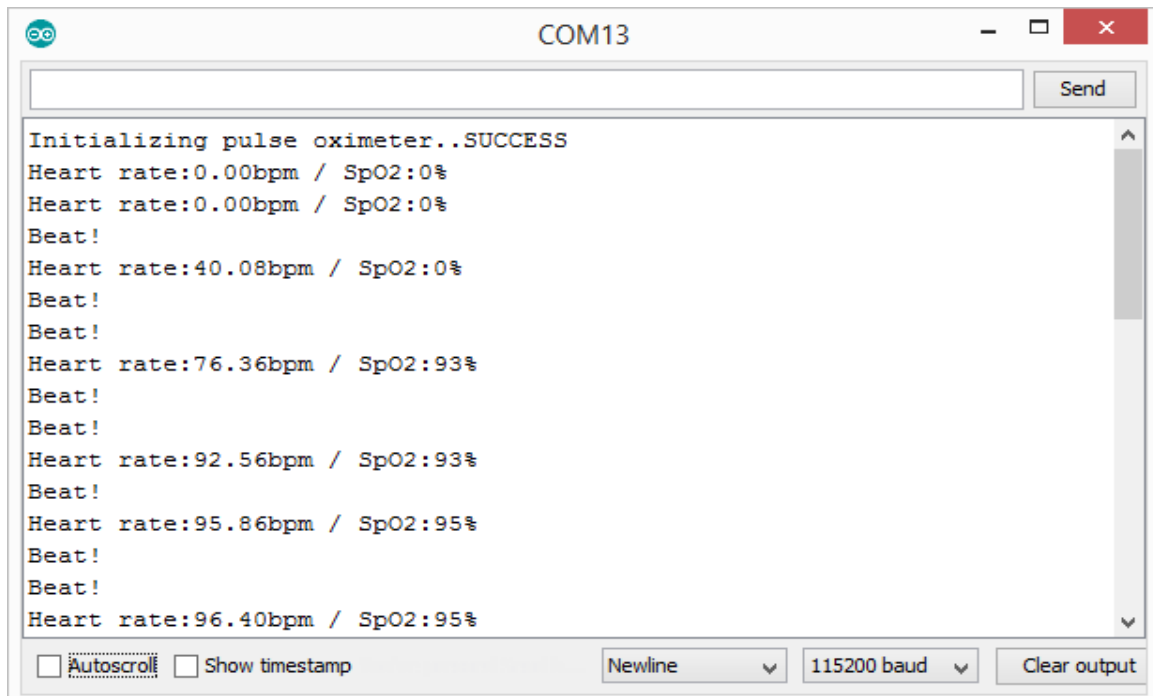


Fig 4.5.4 output readings of spo2 sensor

4.6 Servo Motor:

A **servomechanism**, or **servo**, is an automatic device that uses error-sensing negative feedback to correct the performance of a mechanism. The term correctly applies only to systems where the feedback or error-correction signals help control mechanical position or other parameters. For example, the car's cruise control uses closed loop feedback, which classifies it as a servomechanism.

RC servos are hobbyist remote control devices servos typically employed in radio-controlled models, where they are used to provide actuation for various mechanical systems. Due to their affordability, reliability, and simplicity of control by microprocessors, RC servos are often used in small-scale robotics applications.

RC servos are composed of an electric motor mechanically linked to a potentiometer. A standard RC receiver sends Pulse-width modulation (PWM) signals to the servo. The electronics inside the servo translate the width of the pulse into a position. When the servo is commanded to rotate, the motor is powered until the potentiometer reaches the value corresponding to the commanded position.

RC servos use a three-pin 0.1" spacing jack (female) which mates to standard 0.025" square pins (which should be gold-plated, incidentally). The most common order is Signal, +voltage, ground. The standard voltage is 6VDC, however 4.8V and 12V has also been seen for a few servos. The control signal is a digital PWM signal with a 50Hz frame rate. Within each 20ms timeframe, an active-high digital pulse controls the position. The pulse nominally ranges from 1.0ms to 2.0ms with 1.5ms always being center of range. Pulse widths outside this range can be used for "over travel" -moving the servo beyond its normal range. This PWM signal is sometimes (incorrectly) called Pulse Position Modulation (PPM).

The servo is controlled by three wires: ground, power, and control. The servo will move based on the pulses sent over the control wire, which set the angle of the actuator arm. The servo expects a pulse every 20 ms in order to gain correct information about the angle. The width of the servo pulse dictates the range of the servo's angular motion.

A servo pulse of 1.5 ms width will typically set the servo to its "neutral" position or

45°, a pulse of 1.25 ms could set it to 0° and a pulse of 1.75 ms to 90°. The physical limits and timings of the servo hardware varies between brands and models, but a general servo's angular motion will travel somewhere in the range of 90° - 120° and the neutral position is almost always at 1.5 ms. This is the "standard pulse servo mode" used by all hobby analog servos.

RC servos are usually powered by the receiver which in turn is powered by battery packs or an Electronic speed controller (ESC) with an integrated or a separate Battery eliminator circuit (BEC). Common battery packs are either Ni-Cd, NiMH or lithium-ion polymer battery (Li-Po) type. Voltage ratings vary, but most receivers are operated at 5 V or 6 V.

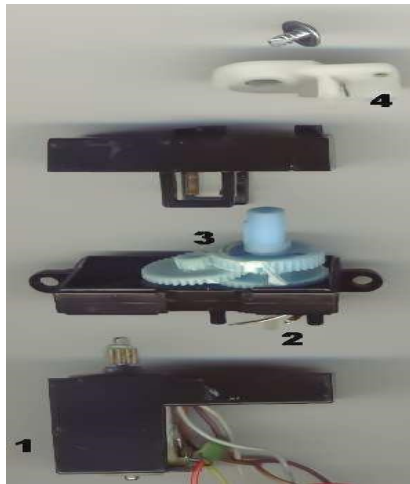


Fig 4.6.1 servo motor

4.7 Buzzer

An audio signaling device like a beeper or buzzer may be electromechanical or piezoelectric or mechanical type. The main function of this is to convert the signal from audio to sound. Generally, it is powered through DC voltage and used in timers, alarm devices, printers, alarms, computers, etc. Based on the various designs, it can generate different sounds like alarm, music, bell & siren.



Fig 4.7.1 buzzer

4.7.1 Buzzer Pin Configuration

The **pin configuration of the buzzer** is shown below. It includes two pins namely positive and negative. The positive terminal of this is represented with the '+' symbol or a longer terminal. This terminal is powered through 6Volts whereas the negative terminal is represented with the '-' symbol or short terminal and it is connected to the GND terminal.

The **specifications of the buzzer** include the following.

- Colour is black
- The frequency range is 3,300Hz
- Operating Temperature ranges from – 20° C to +60°C
- Operating voltage ranges from 3V to 24V DC
- The sound pressure level is 85dBA or 10cm
- The supply current is below 15mA

Types of Buzzer

A buzzer is available in different types which include the following.

- Piezoelectric
- Electromagnetic

- Mechanical
- Electromechanical
- Magnetic
-

4.7.2 Working Principle

The working principle of a buzzer depends on the theory that, once the voltage is given across a piezoelectric material, then a pressure difference is produced. A piezo type includes piezo crystals among two conductors.

Once a potential disparity is given across these crystals, then they thrust one conductor & drag the additional conductor through their internal property. So this continuous action will produce a sharp sound signal.

The mounting configurations of buzzers include the following.

- Panel Mount
- Wire Leads
- Screw Terminals
- Through Hole
- Spring Contact
- Surface Mount

Advantages

The **advantages of a buzzer** include the following.

- Simply Compatible
- Frequency Response is Good
- Size is small
- Energy Consumption is less
- The Range of Voltage usage is Large
- Sound Pressure is high

Disadvantages

The **disadvantages of the buzzer** include the following.

- Controlling is a little hard
- Generates Annoying Sound
- Training is necessary to know how to repair the condition without just turning off.

4.7.3 Applications

The **applications of the buzzer** include the following.

- Communication Devices
- Electronics used in Automobiles
- Alarm Circuits
- Portable Devices
- Security Systems
- Timers
- Household Appliances
- Electronic Metronomes
- Sporting Events
- Annunciator Panels
- Game Shows

Thus, this is all about an overview of a buzzer data sheet that includes its working principle, pin configuration, specifications, circuit, advantages, disadvantages & its applications. It is an electromechanical, electromagnetic, mechanical, piezoelectric, electro-acoustic audio signaling device. This buzzer works through an audio signal source or oscillating circuit. A ring or beep or click indicates that a switch has been pushed.

4.8 BMP 180 sensor

The BMP180 (Barometric Pressure/Temperature sensor) is the new digital barometric pressure sensor of Bosch Sensortec, with a very high performance, which enables applications in advanced mobile devices, such as smart phones, tablet PCs and sports devices

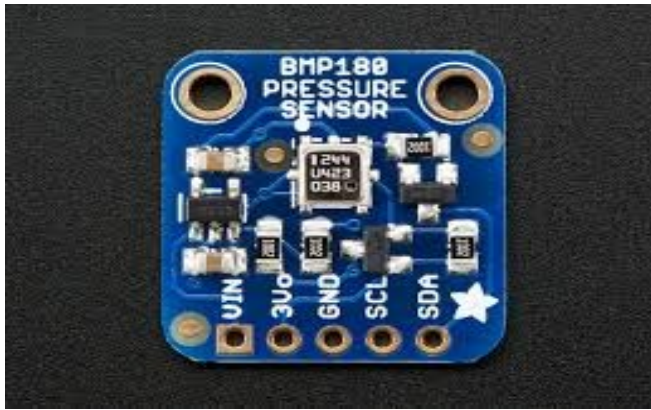


Fig 4.8.1 bmp pressure sensor

The BMP180 sensor is mainly used to measure atmospheric pressure or biometric pressure. The working principle of the air pressure sensor is very simple, it works based on the weight of air. Because the air around us has a certain weight, and this weight has a specific pressure.

This precision sensor from Bosch is the best low-cost sensing solution for measuring barometric pressure and temperature. Because pressure changes with altitude you can also use it as an altimeter! The sensor is soldered onto a PCB with a 3.3V regulator, I2C level shifter and pull-up resistors on the I2C pins.

4.8.1 Sensor pinout

- VCC - Connected to +5V
- GND - Connected to ground.
- SDA - Serial Data pin (I2C interface)
- SCL - Serial Clock pin (I2C interface)
- VO - If +5V is not present, we can connect to +3.3 v.

4.8.2 Sensor specification

- Operating voltage of BMP180: 1.3V – 3.6V
- Input voltage of BMP180MODULE: 3.3V to 5.5V
- Peak current: 1000uA
- Consumes 0.1uA standby
- Maximum voltage at SDA, SCL : VCC + 0.3V
- Operating temperature: -40°C to +80°.

CHAPTER 5

RESULT

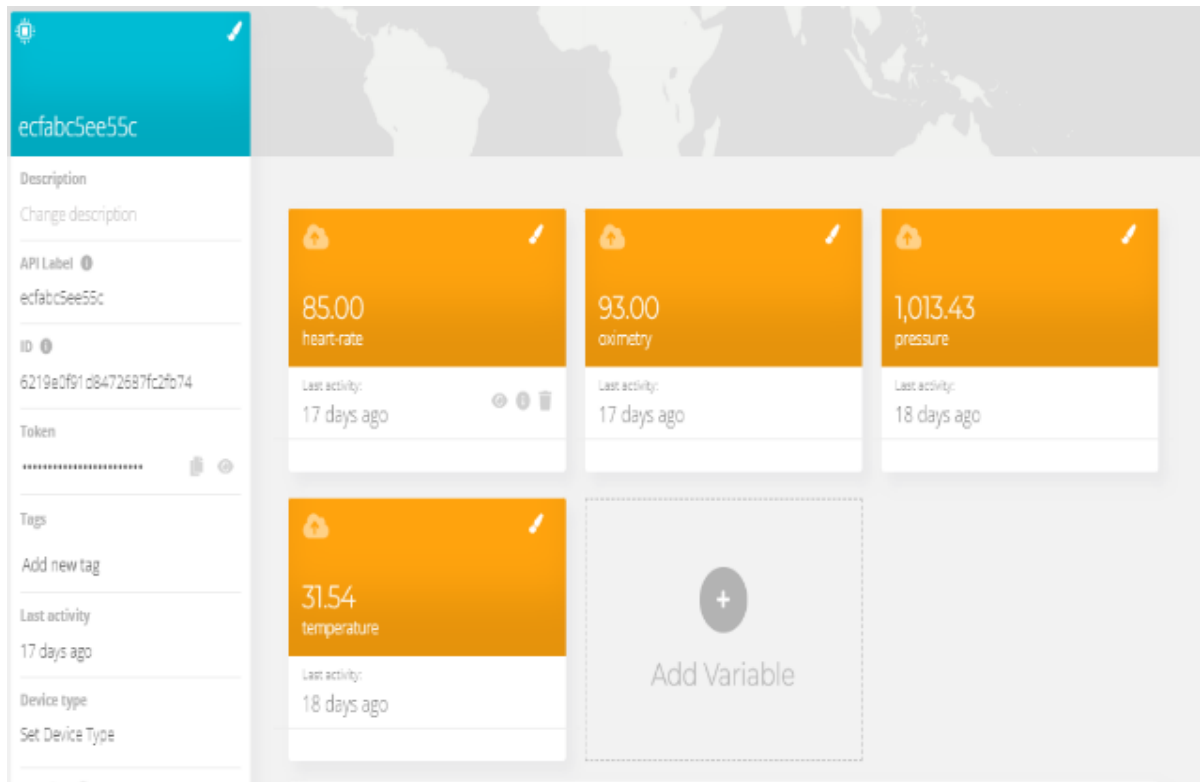


FIG 5.1 UBIDOTS RESULT

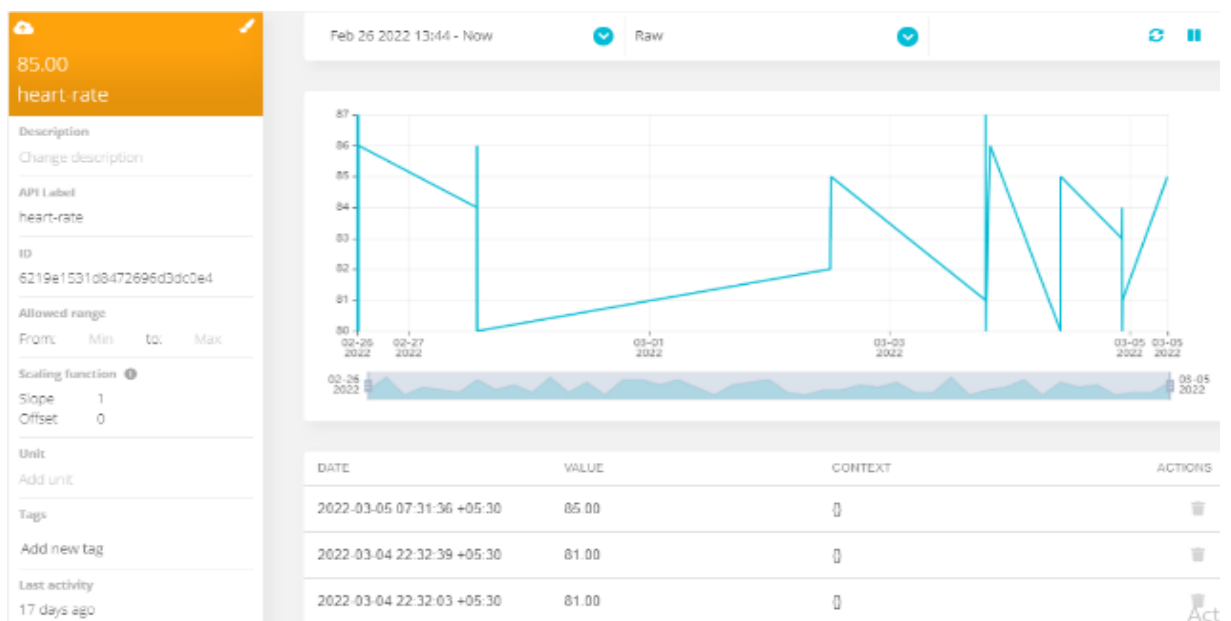


FIG 5.2 HEARTBEAT VALUES

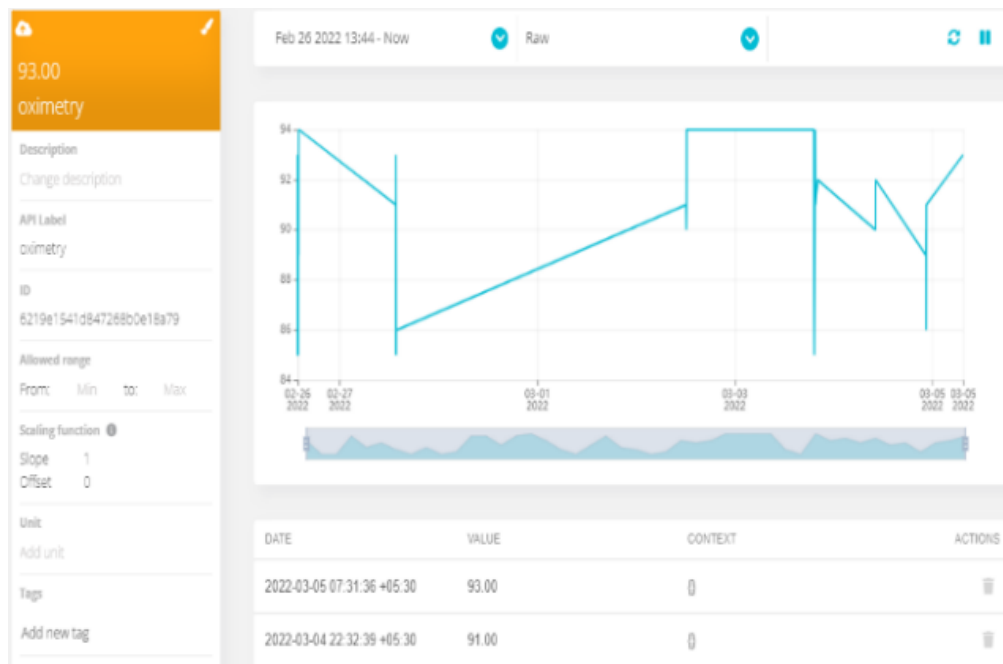


FIG 5.3 OXIMETRY VALUES

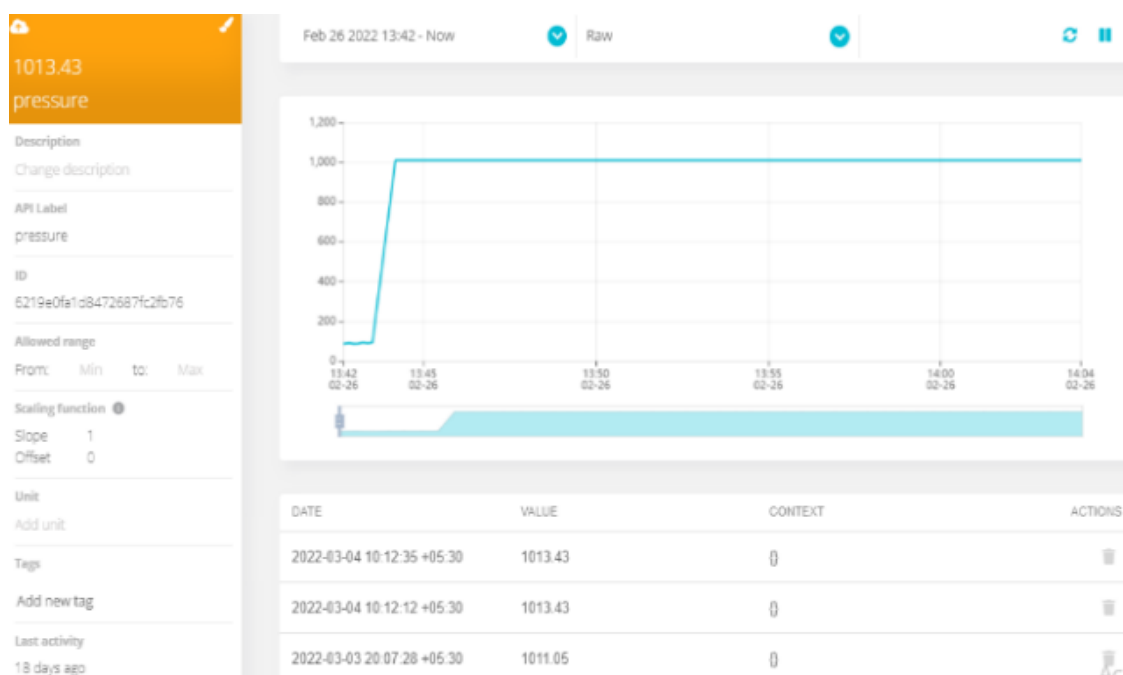


FIG 5.4 PRESSURE SENSOR VALUES

CHAPTER 6

SUMMARY AND CONCLUSION

By using this system, we can get the patient's health condition. Especially blood pressure measurement. If BPM varies then automatically ventilation will vary according to those values. This value can be uploaded into the cloud. From cloud we can get the alert messages as well as gmail alert. And also, patient heartbeat and oxygen level can be measured

REFERENCES

- [1] Vandana Milind Rohokale, Neeli Rashmi Prasad, Ramjee Prasad “A Cooperative Internet of Things (IoT) for Rural Healthcare Monitoring and Control” 2011 Center for TeleInfrastruktur, Aalborg University, Denmark, P.P 978-1-4577-0787-2/11.
- [2] Charalampos dukes, Ilias Maglogiannis “Bringing IoT and Cloud Computing towards Pervasive Healthcare” 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, P.P 978-0-7695-4684-1/12.
- [3] Junaid Mohammed, Abhinav Thakral, Adrian Filipb Ocneanu, Colin Jones, Chung-Horng Lung, Andy Adler “Internet of Things: Remote Patient Monitoring Using Web Services and Cloud Computing” 2014 IEEE International Conference on Internet of Things (iThings 2014), Green Computing and Communications (GreenCom2014), and Cyber-Physical-Social Computing (CPSCoM 2014),P.P 978 1-4799-5967-9/14.
- [4] Tae-Yoon Kim, Sungkwan Youm, Jai-Jin Jung, Eui-Jik Kim “Multi-hop WBAN Construction for Healthcare IoT” 2015 International Conference on Platform Technology and Service, P.P 978-1-4799-1888-1/15.
- [5] Boyi Xu, Li Da Xu, Senior Member, IEEE, Hongming Cai, Cheng Xie, Jingyuan Hu, and Fenglin Bu “Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services” IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 10, NO. 2, MAY 2014, P.P 1551-3203.
- [6] Joseph Bailey, Michael Fecteau, Noah L. Pendleton “Wireless Pulse Oximeter” 2017
- [7] Karsten Aalders & Charles R. Abbruscato “Device specialization Pulse oximeter” 2021