CHAPTER-1
INTRODUCTION

1.1 INTRODUCTION

In today's dynamic healthcare landscape, IoT-based Patient Health Monitoring Systems are redefining the standard of care. Built upon a foundation of cutting-edge technology, these systems integrate components such as Arduino, heartbeat sensors, MEMS sensors, temperature sensors, GSM modules, GPS modules, and NodeMCU platforms to provide comprehensive monitoring and management of patient health. Arduino serves as the central hub, orchestrating the integration of various sensors and facilitating data processing, while heartbeat sensors offer real-time monitoring of cardiac activity, enabling early detection of abnormalities. MEMS sensors complement this by capturing motion, orientation, and environmental data, providing a holistic view of patient well-being. Temperature sensors contribute to monitoring physiological parameters, aiding in the diagnosis and management of conditions such as fever or hypothermia.

Additionally, GSM and GPS modules enable communication and location tracking, ensuring timely response in emergencies. NodeMCU, with its capabilities for wireless connectivity and cloud integration, acts as the gateway for data transmission, allowing healthcare professionals to access patient data remotely. The integration of these components creates a powerful ecosystem that empowers healthcare providers with actionable insights, enhances patient engagement, and ultimately improves outcomes. Through continuous monitoring and analysis of vital signs, IoT-based systems enable early intervention, preventing complications and improving quality of life for patients. Furthermore, these systems promote patient autonomy by enabling remote monitoring, allowing individuals to manage their health from the comfort of their homes.

1.2 AIM OF THE PROJECT

Here the main objective is to design a Remote Patient Health Monitoring System to diagnose the health condition of the patients. Giving care and health assistance to the bedridden patients at critical stages with advanced medical facilities have become one of the major problems in the modern hectic world. In hospitals where many patients whose physical conditions must be monitored frequently as a part of a diagnostic procedure, the need for a cost-effective and fast responding alert mechanism is inevitable.

1.3 SIGNIFICANCE OF THE WORK & APPLICATIONS

IoT (Internet of Things) based patient health monitoring systems offer a revolutionary approach to healthcare by integrating technology into the monitoring and management of patients' health in real-time. Here are some of their significance and applications:

- **Real-Time Monitoring:** IoT devices allow continuous and real-time monitoring of vital signs such as heart rate, blood pressure, temperature, and more. This enables healthcare providers to have immediate access to patient data and respond promptly to any anomalies or emergencies.
- **Remote Patient Monitoring:** Patients can be monitored remotely from their homes, reducing the need for frequent hospital visits. This is particularly beneficial for patients with chronic conditions, elderly patients, or those recovering from surgery who require constant monitoring but do not need to be hospitalized.
- **Early Detection of Issues:** By continuously monitoring vital signs and other health parameters, IoT systems can detect health issues at an early stage, allowing for timely intervention and preventing complications. This proactive approach can significantly improve patient outcomes and reduce healthcare costs.
- **Personalized Medicine:** IoT-based patient health monitoring systems can collect large amounts of patient data over time. By analyzing this data using artificial intelligence and machine learning algorithms,

healthcare providers can gain insights into individual health trends and tailor treatment plans according to each patient's unique needs and characteristics.

- **Emergency Response:** IoT devices can automatically alert healthcare providers or emergency services in case of critical health events such as heart attacks, seizures, or falls. This ensures that patients receive prompt medical assistance, potentially saving lives in emergency situations.

CHAPTER-2
LITERATURE SURVEY

2.1 LITERATURE SURVEY

Ananda Mohan Ghosh et al. has proposed a health monitoring system for managing the hospital to allow family members and consultant doctors to remotely monitoring the patient's health condition through the internet with E-health sensor shield kit interface kit. But it does not send any notification such as email and SMS alert to the respective family members and doctors.

P Kumar et al. has proposed a patient monitoring healthcare system which is controlled by a raspberry pi such as the heartbeat rate, respiration level, and temperature and body movement of the patient is monitored and data is collected by using sensors and displayed it on the screen using the putty software. However, it does not provide the alarm notification for insisting the family members or doctors give the prescribed drugs to the patient which is included in our proposed solution.

Sarfraz Fayaz Khan has demonstrated a useful patient’s healthcare monitoring system with the help of IoT and RFID tags. But it does not contain preventive measures concerning the patient health condition by controlling the appliances and providing the prescribed drugs to the patient which is included in our paper.

Freddy Jimenez et al. have considered only on monitoring the patient’s health condition and sending the necessary information and notification to doctors, family members. Moreover, it does not contain the appliance control, which is included in our project; it only focused on Monitoring.

S. Siva et al. has demonstrated to monitor a patient’s health condition by using the smart hospital system. The health condition of the patients can be monitored by using the spark kit. It gathers information about the temperature and heartbeat rate of the patient and sent an alert notification if any of the obtained parameters crosses the predefined threshold value.

2.2 EXISTING SYSTEM

Before the advent of modern IoT-based patient health monitoring systems, healthcare monitoring was primarily conducted using traditional methods, which often involved manual data collection and limited remote monitoring capabilities. Here's an overview of the old system:

- **Manual Vital Sign Monitoring:** In the past, vital signs such as heart rate, blood pressure, temperature, and oxygen saturation were typically monitored manually by healthcare professionals using standalone medical devices. Patients often had to visit healthcare facilities regularly for check-ups, and vital signs were recorded during these visits.
- **Hospital Based Monitoring Systems:** Patients requiring continuous monitoring were typically admitted to hospitals or healthcare facilities where specialized monitoring equipment was available. These monitoring systems were often stationary and confined to hospital settings, limiting patients' mobility and independence.
- **Limited Remote Monitoring:** Remote monitoring capabilities were limited, and patients had to rely on periodic check-ups or telephone consultations with healthcare providers for remote support. There was no real-time transmission of patient data, and healthcare providers had limited visibility into patients' health status outside of healthcare facilities.

2.3 PROPOSED SYSTEM

The system has fundamental concept of continuously monitoring patient's health condition through online. As it's effect, for this health monitoring system design three stages of architectural elements occurs, they are: (1) Sensor unit (2) Data dealing out Module (3) Web User boundary. The different sensors can interface and can used to get the data from the patient's body through physiological indications. Then the collected data is uploaded using an ESP8266 module before that data's being sent to the IoT web server. Thing Speak is used for the graphical reading and to display the collected data in the web user boundary. The present readings and process of transactions are displayed through Thing Speak platform. For the communication between web server and Wi-Fi module HTTP protocol; can be used. Patients can be monitored in real time thanks to the user interface. We can see the system architecture for developed system in figure 1. The different sensors are together employed to get the data from the hospital environment, Patients will have their pulse rate and SpO2 measured using a max30100 sensor, temperature measured with an LM35 sensor, blood pressure measured with a BMP280 sensor, and measurement results displayed on a mobile app and LCD display. All the sensors are coupled to the ESP8266 computing point. When all those (temperature, heartbeat, blood pressure, and blood oxygen level) sensors are interfaced, nodeMCU becomes the heart of the system. The signals are detected by the sensors and detectors in analogue format, which must then be converted to digital format. The information is transferred to a cloud or server. The ESP8266 captures sensor data and sends it wirelessly to IoT websites. The data will be shown in the mobile app using a bluetooth module that receives data from the nodeMCU and saves it to the cloud. The board has its own processing unit and Wi-Fi. The output of the sensors is then interfaced to the IoT web server. The information will be accessed by any network-enabled devices. The data will be plotted graphically, and we have channel based system. Then the data will be processed using an ESP8266 module before being sent to the gateway server. In the web user interface, for graphical interpretation and to display the collected data we can utilize Thing Speak platform . The present status and process of transactions are displayed through Thing Speak. On the LCD panel, calculated data from the human being can also be shown.

CHAPTER-3
IMPLEMENTATION

3.1 BLOCK DIAGRAM

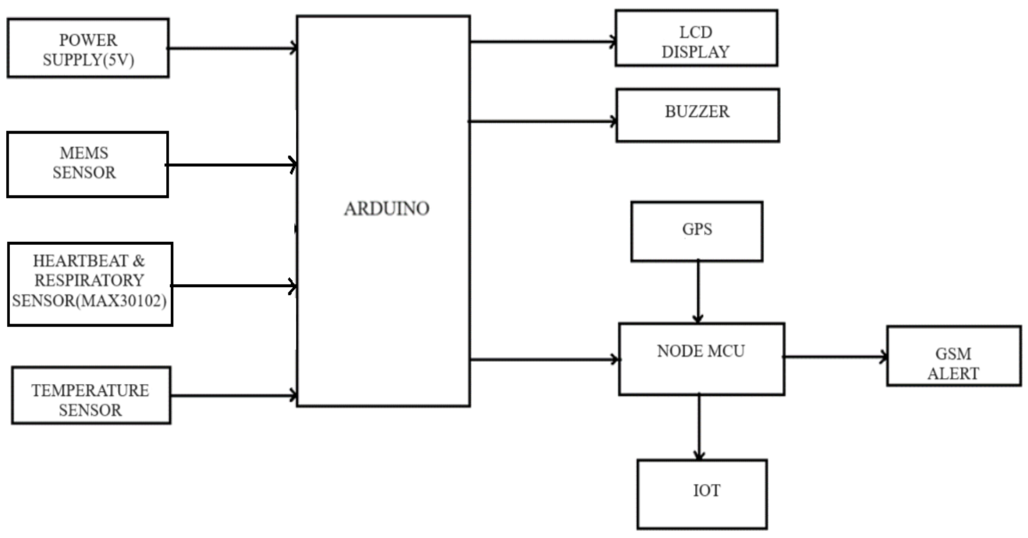


Fig 3.1: Block Diagram

The block diagram of our IoT-based patient health monitoring system consists of several interconnected components that work together to ensure continuous health monitoring and real-time data reporting. The whole system runs with power supply that provides the necessary energy to all components, ensuring uninterrupted functionality. The system includes various sensors: a MEMS (Micro-Electro-Mechanical Systems) sensor for detecting movement and orientation, a MAX30102 sensor for measuring heart rate and SpO2 (blood oxygen levels), and a temperature sensor for monitoring body temperature. These sensors continuously collect vital health data from the patient. An LCD display is used to show real-time readings locally, allowing the patient and nearby caregivers to easily view the health metrics.

Data from these sensors is sent to the NodeMCU, a microcontroller with built-in Wi-Fi capabilities, which processes and transmits the information to the cloud via the IoT (Internet of Things) network. This setup enables remote monitoring by healthcare providers. Additionally, the system includes a GPS module to track the patient's location, which is essential in emergencies or for monitoring patients with mobility issues. A GSM module is incorporated to ensure data and alerts can be sent over cellular networks when Wi-Fi is unavailable. To provide immediate local alerts, a buzzer is included, which sounds an alarm if any health parameters exceed predefined safety thresholds. This comprehensive system leverages IoT technology to offer continuous, real-time health monitoring, enhancing patient care and safety.

3.2 SCHEMATIC DIAGRAM

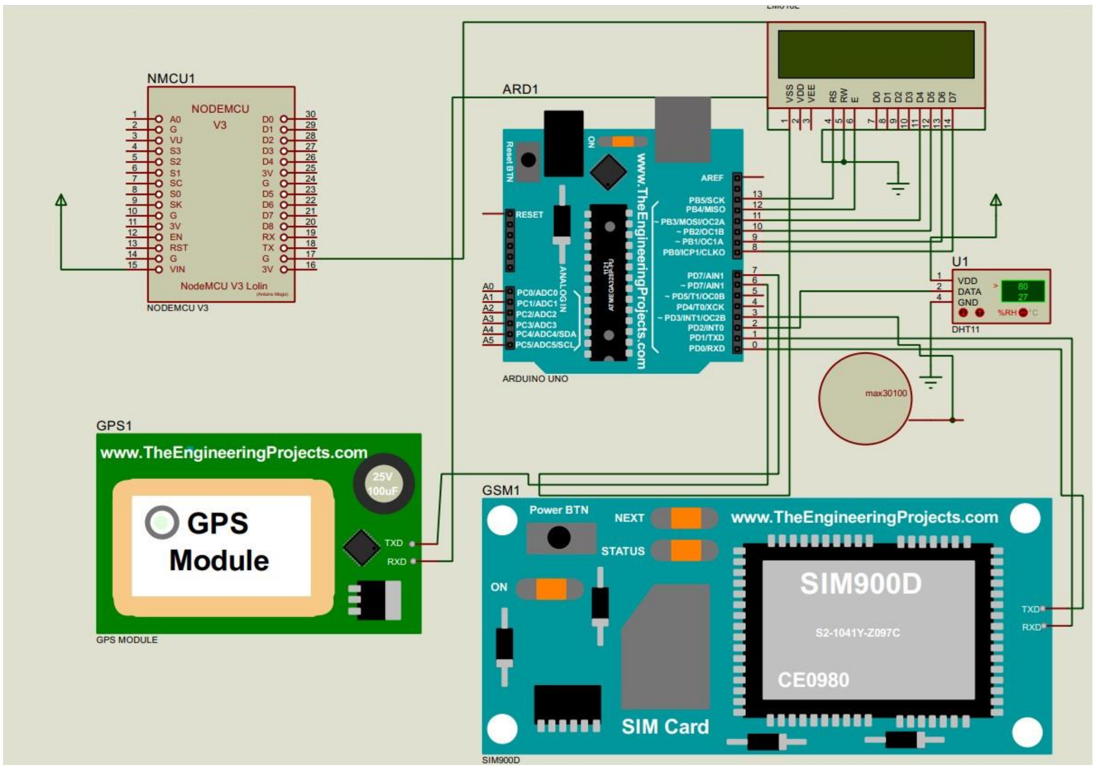


Fig 3.2: Schematic Diagram

Arduino is externally interfaced with the Liquid Crystal Display, digital pins 8, 9, 10, 11, 12, 13 are connected to D7, D6, D5, D4, Enable and Register Select pins respectively. Digital pins 0 and 1 of Arduino is connected to the TXD and RXD pins of GSM(SIM900D) respectively.

Digital pin 3 of Arduino is connected to the Vin pin of MAX30100 Heart Beat Sensor and the digital pin 2 is connected to the DATA pin of DHT11 (Temperature Sensor).

TXD(Transmit Data) pin of GPS module is connected to the VSS pin of D6 pin of Arduino and RXD(Receive Data) pin is connected to the VSS of Liquide Crystal Display.

Ground pin of Node MCU is connected to the RW(Read/Write) pin of LCD and both together are grounded at a point.

Buzzer is an external output device which is connected to the Analog Pin0 (A0) of Arduino.

3.3 WORKING PRINCIPLE

In this project, detecting the various parameters of the patient using Internet of Things is done. In health monitoring system based on IoT projects, the real time factors of the patient are sent to the cloud by using internet connection. These data can be sent to anywhere in the world, so that the user will view the details anytime. This is the major advantage over SMS based health monitoring system. In IoT based patient health monitoring system, data of the patient health are often seen by doctors. The reason behind is that, the data has to be accessed by visiting an internet site or computer address, whereas in Global System for Mobile communication based patient monitoring system, the health parameters are sent using GSM via SMS. IoT based health monitoring system has 3 senses. Initial one is a temperature sensor, second one is heartbeat device, and the third one is respiratory sensor. This is extremely useful since the doctor will detect the patient’s health parameters simply by visiting internet website or IP address. And today several IoT apps are also being developed. So, the doctor and relatives will monitor or track the patient health through Android apps. To operate an IoT based health tracking system, you will need a Wi-Fi connection. The microcontroller or the Arduino board connects to the Wi-Fi network using Wi-Fi module. This system will not work when there is no Wi-Fi network, Arduino UNO board continually reads the input from 3 senses. Then it sends the information to the cloud by sending this information to specific URL/IP address. Then this action of accelerating data to the cloud is repeated for a specific period of time.

CHAPTER 4
HARDWARE DESCRIPTION

4.1 ARDUINO UNO

The Arduino Uno is a microcontroller board based on the ATmega328 It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to serial converter. "Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

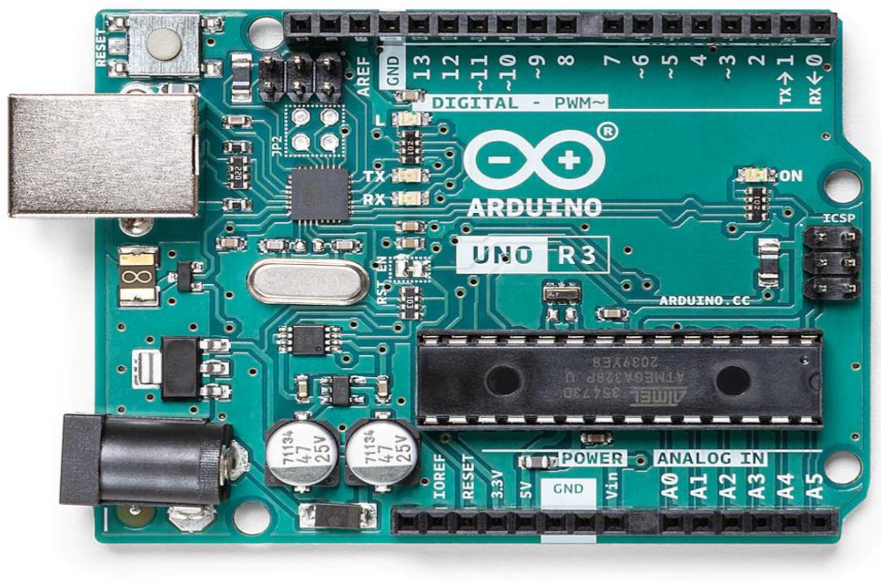


Fig 4.1: Arduino UNO

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.

- **ATmega328p Microcontroller-** It is a single chip Microcontroller of the ATMEL family. The processor code inside it is of 8-bit. It combines Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timer, external and internal interrupts, and oscillator.
- **ICSP pin -** The In-Circuit Serial Programming pin allows the user to program using the firmware of the Arduino board.
- **Power LED Indicator-** The ON status of LED shows the power is activated. When the power is OFF, the LED will not light up.
- **Digital I/O pins-** The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
- **TX and RX LED's-** The successful flow of data is represented by the lighting of these LED's.
- **AREF-** The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
- **Reset button-** It is used to add a Reset button to the connection.
- **USB-** It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
- **Crystal Oscillator-** The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.
- **Voltage Regulator-** The voltage regulator converts the input voltage to 5V.
- **GND-** Ground pins. The ground pin acts as a pin with zero voltage.
- **Vin-** It is the input voltage.
- **Analog Pins-** The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pins.

Arduino Uno is an open-source microcontroller board based on the processor ATmega328P. There are 14 digital I/O pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains all the necessary modules needed to support the microcontroller. Just plug it into a computer with a USB cable or power it with an AC-to-DC adapter to get started. In the event of a worst-case scenario, one could replace it with a new one as the Uno is very economical compared to other boards like raspberry pi, STM, etc. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega1612 (Atmega81/2 up to version R2) programmed as a USB- to-serial converters.

ATmega328P is a very advance and feature rich microcontroller. It is one of a famous microcontroller of Atmel because of its use in ARDUINO UNO board. It is a microcontroller from the Atmel’s mega MVR microcontrollers family (Later in 2016 the Atmel is obtained by Microchip Technology Inc, the microcontrollers manufactured in mega MVR family are designed for handling larger program memories and each microcontroller in this family contains different amount of ROM, RAM, I/O pins and other features and also, they are manufactured in different output pins which are from 8 pins to hundreds of pins.

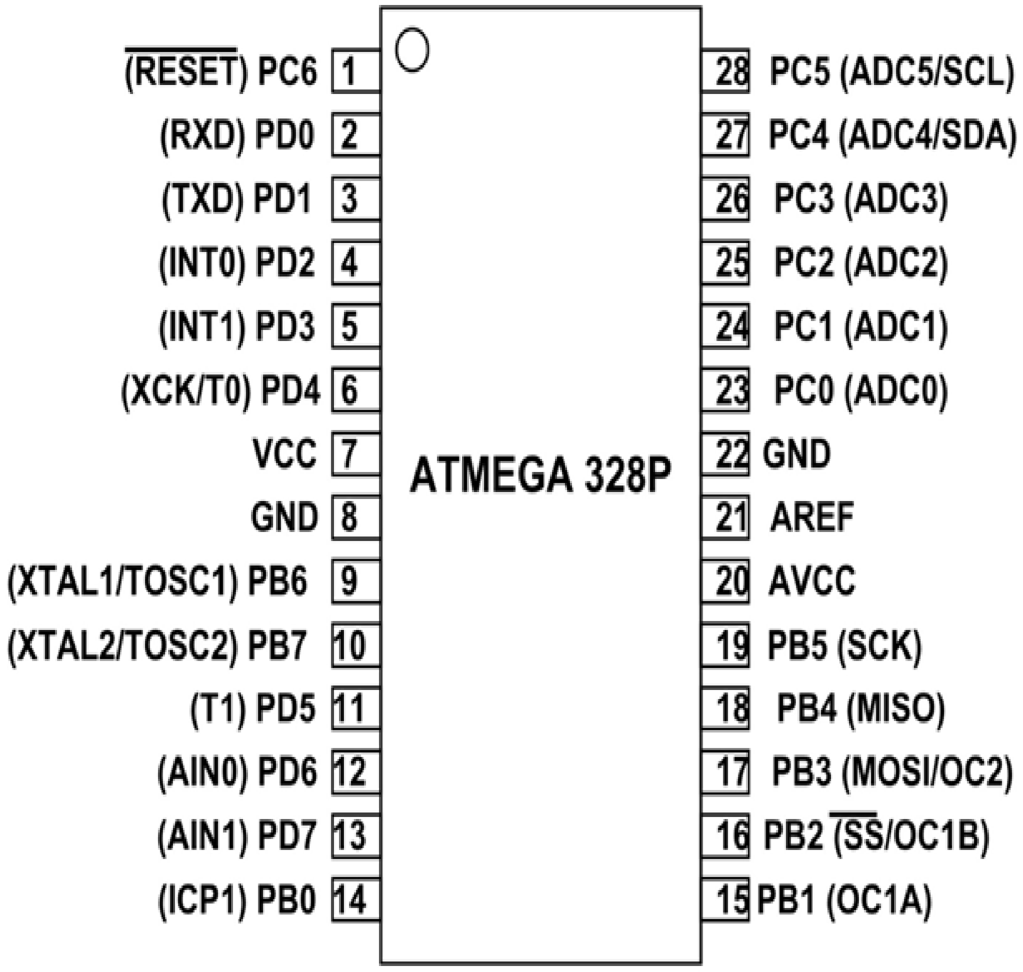


Fig 4.2: Pin Diagram

The internal circuitry of **ATmega328P** is designed with low current consumption features. The chip contains 32 kilobytes of internal flash memory, 1 kilo bytes of EEPROM and 2 kilo-bytes of SRAM. The EEPROM and the flash memory are the memories which saves information and that information still exits every-the power is disconnected or off but the SRAM is a memory which only saves the information until the power is supplied and when the power is disconnected all the information saved in SRAM will be erased.

4.1.1 SPECIFICATIONS AND FEATURES

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V

- In-out Voltage (limit): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- PWM Digital I/O Pins: 6
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 Ma
- DC current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- Clock Speed: 16 MHz
- LED_BUILTIN: 13
- Length: 68.6 mm
- Width: 58.4 mm

4.2 NODE MCU

The **NodeMCU ESP8266 development board** comes with the ESP-12E module containing the ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating features make it ideal for IoT projects.

NodeMCU can be powered using a Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.

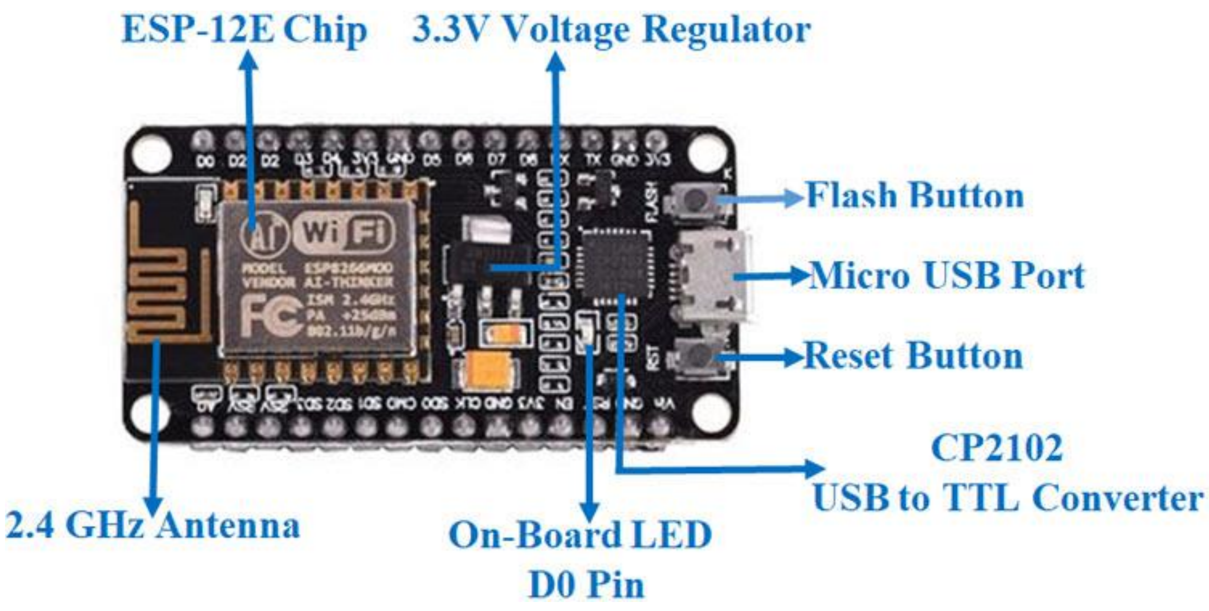


Fig 4.3: Node MCU

4.2.1 SPECIFICATIONS AND FEATURES

- Operating Voltage: 3.3V
- Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
- Input Voltage: 7-12V
- Digital I/O Pins (DIO): 16
- Analog Input Pins (ADC): 1
- UARTs: 1
- SPIs: 1
- I2Cs: 1
- Flash Memory: 4 MB
- SRAM: 64 KB
- Clock Speed: 80 MHz
- USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
- PCB Antenna
- Small Sized module to fit smartly inside your IoT projects

4.3 MEMS SENSOR

MEMS or Micro-Electro-Mechanical Systems, is a technology that may be characterized as miniature mechanical and electrical parts (i.e., devices and structures) that are created using micro fabrication processes. MEMS devise key 15 physical dimensions range from far below one micron on the lowest of the dimension spectrum to several millimeters on the high end. MEMS devices range in complexity from basic structures with no moving parts to very sophisticated electromechanical systems with several moving elements controlled by integrated microelectronics. The essential condition for MEMS is that at minimum some of the components have some type of mechanical functioning, regardless of whether or not they can move. In different regions of the globe, MEMS is referred to by different names.

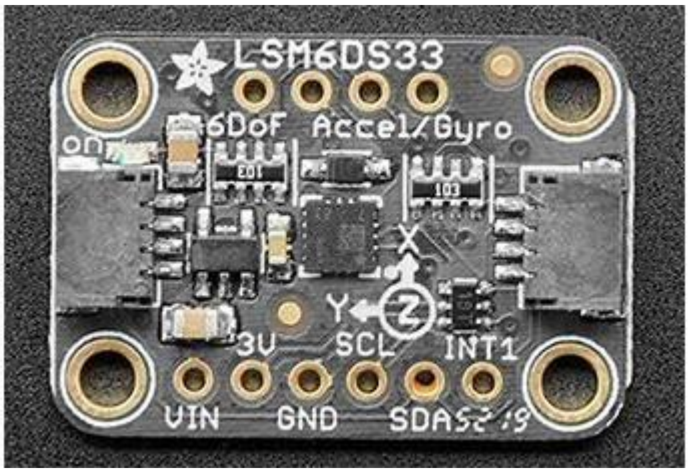


Fig 4.4: MEMS Sensor

4.3.1 SPECIFICATIONS AND FEATURES

- Supply voltage: 3.3 V to 5V
- TTL level output
- Maximum output current: 15mA
- Can work on low voltages
- Maximum operating temperature: 0°C to + 80°C
- Easy interface
- Long life.

4.4 HEARTBEAT & RESPIRATORY SENSOR

The sound of a person's heartbeat is the sound of the heart's valves contracting or expanding as they drive blood from one region to another. The heartbeat rate is the number of times the heartbeat per minute (BPM), and the pulse is the heartbeat that can be felt in any artery adjacent to the skin.

Manual Method: Checking one's pulses at two locations—the wrists (the radial pulse) as well as the neck—can be used to assess one's heartbeat (carotid pulse). The process is to lay a finger (index and right hand) on the wrist (or neck just below the throat) and count the number of pulses for 30 seconds, then multiply the result by two to get the cardiac rate. Nonetheless, pressure must be used carefully, and fingers should be shifted down and up until the pulse is sensed.

Using a sensor: The optical power variation as light is reflected or absorbed in its whole route through the blood as the heartbeat changes are being used to measure heart rate.

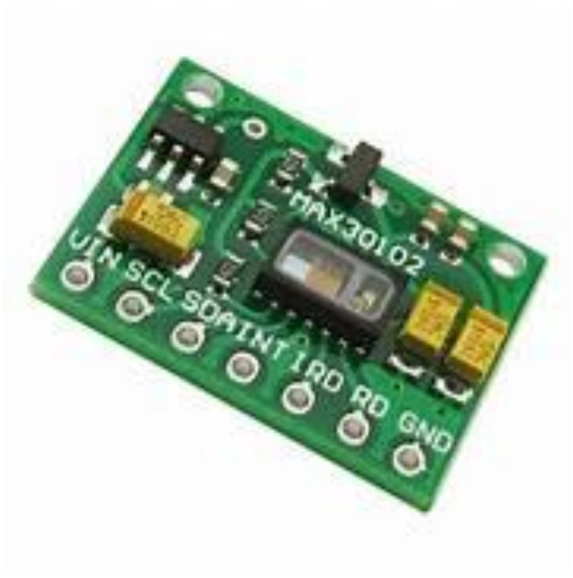


Fig 4.5: MAX30102 Sensor

MAX30102 is a multipurpose sensor used for multiple applications. It is a **heart rate monitoring** sensor along with a **pulse oximeter**. The sensor comprises two Light Emitting Diodes, a photodetector, and a series of low noise signal processing devices to detect heart rate and to perform pulse oximetry.

4.4.1 SPECIFICATIONS AND FEATURES OF MAX30102

- Operating Voltage - 1.8V to 3.3V
- Input Current - 20Ma
- Integrated Ambient Light Cancellation
- High Sample Rate Capability
- Fast Data Output Capability.

4.5 TEMPERATURE SENSOR

A temperature sensor is an electrical device that gives temperature measurement in a readable form, such as a Thermocouple or RTD (Resistive Temperature Detector). The coolness or hotness of an entity is measured by the temperature sensor. The voltage read across the diode is what makes the sensor work. When the voltage is increased, the temperature rises, and the voltage between the emitter and base terminals of the transistor decreases. The sensor has saved that information. The gadget generates an analogue signal when the difference in voltage is amplified, and it is directly proportional to the temperature. Temperature sensors come in a variety of shapes and sizes.

4.5.1 SPECIFICATIONS AND FEATURES OF DHT11

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

Fig 4.6: Temperature Sensor

4.6 GPS MODULE

"Global Positioning System" is the abbreviation for "GPS". The Global Positioning System (GPS) is a satellite navigation system that determines an object's ground position. The United States military initially deployed GPS technology in the 1960s, and civilian adoption followed over the next few decades. Many commercial items, such as autos, cell phones, workout watches, and GPS Devices, now contain GPS receivers. The GPS has almost 24 satellites floating in space around 12,000 miles (19,300 km) above the surface. They orbit the heart every 12 hours at a speed of around 7,000 km (11,200kilometers per hour). The satellites are equally spaced, providing direct route access to four of them from everyone on the planet. Each GPS satellite sends out a message with the spacecraft's present state, orbit, and precise time. A GPS receiver uses triangulation to estimate its exact position by combining signals from several satellites. To calculate a receiver's position, three satellites are necessary, while a link to satellites is optimal because it gives more precision.

Fig 4.7: GPS Module

4.6.1 SPECIFICATIONS AND FEATURES OF GPS MODULE

- Supply voltage: 3.6V
- Maximum DC current at any output: 10mA
- Operation limits: Gravity-4g, Altitude-50000m, Velocity-500m/s
- Operating temperature range: -40ºC TO 85°C

4.7 GSM MODULE

GSM is a digitized mobile network commonly utilized by phone users in Europe and other areas of the world. GSM is the most extensively exploited of the three wireless mobile telephony technologies: time division multiple access (TDMA), code division multiple access (CDMA) (CDMA), and GSM. GSM serializes and compresses data before transmitting it together with two additional streams of user information, each in its time slot, via a channel. It works in the 900 MHz or 1,800 MHz frequency bands. GSM, along with other wireless mobile communications technology, is key to the development of wireless mobile telecommunications, which includes High-Speed Circuit-Switched Data (HSCSD), General Packet Radio Service (GPRS), Enhanced Data GSM Environment (EDGE), and Universal Mobile Telecommunications Provider (UMTS).

4.7.1 FEATURES OF GSM

- Supports international roaming
- Clear voice clarity
- Ability to support multiple handheld devices.
- Spectral / frequency efficiency
- Low powered handheld devices.
- Ease of accessing network
- International ISDN compatibility.
- Low service cost.
- New features and services.

4.7.2 SPECIFICATIONS OF GSM

- Frequency band—The frequency range specified for GSM is 1,850 to 1,990 MHz (mobile station to base station).
- Duplex distance—The duplex distance is 80 MHz. Duplex distance is the distance between the uplink and downlink frequencies. A channel has two frequencies, 80 MHz apart.
- Channel separation—The separation between adjacent carrier frequencies. In GSM, this is 200 kHz.
- Modulation—Modulation is the process of sending a signal by changing the characteristics of a carrier frequency. This is done in GSM
  via Gaussian minimum shift keying (GMSK).
- Transmission rate—GSM is a digital system with an over-the-air bit rate of 270 kbps.
- Access method—GSM utilizes the time division multiple access (TDMA) concept. TDMA is a technique in which several different calls
  may share the same carrier. Each call is assigned a particular time slot.
- Speech coder—GSM uses linear predictive coding (LPC). The purpose of LPC is to reduce the bit rate. The LPC provides parameters for a filter that mimics the vocal tract. The signal passes through this filter, leaving behind a residual signal. Speech is encoded at 13 kbps.

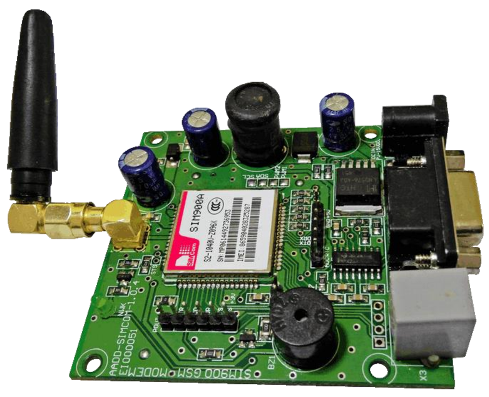


Fig 4.8: GSM Module

4.8 LCD DISPLAY (LIQUID CRYSTAL DISPLAY)

The term LCD stands for liquid crystal display. It is one kind of electronic display module used in an extensive range of applications like various circuits & devices like mobile phones, calculators, computers, TV sets, etc. These displays are mainly preferred for multi-segment light- emitting diodes and seven segments. The main benefits of using this module are inexpensive; simply programmable, animations, and there are no limitations for displaying custom characters, special and even animations, etc.

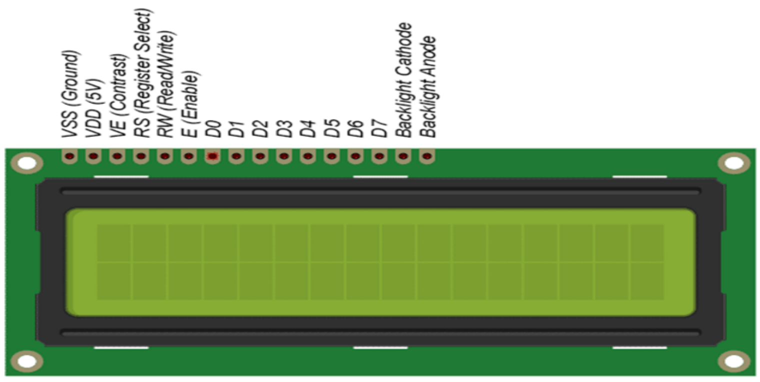
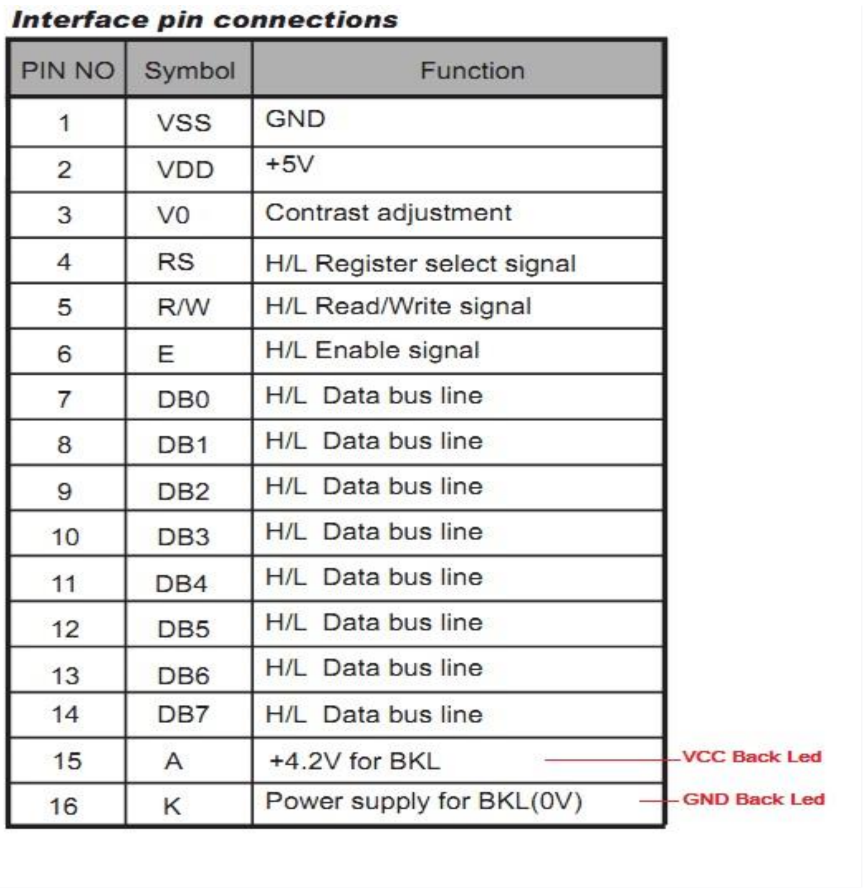


Fig 4.9: Liquid Crystal Display

A 16×2 LCD has two registers like data register and command register. The RS (register select) is mainly used to change from one register to another. When the register set is ‘0’, then it is known as command register. Similarly, when the register set is ‘1’, then it is known as data register.

4.8.1 PIN CONFIGURATION OF LCD



Interface pin connections

PIN NO	Symbol	Function
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A	+4.2V for BKL
16	K	Power supply for BKL(0V)

Table-4.1: Pin configuration of LCD

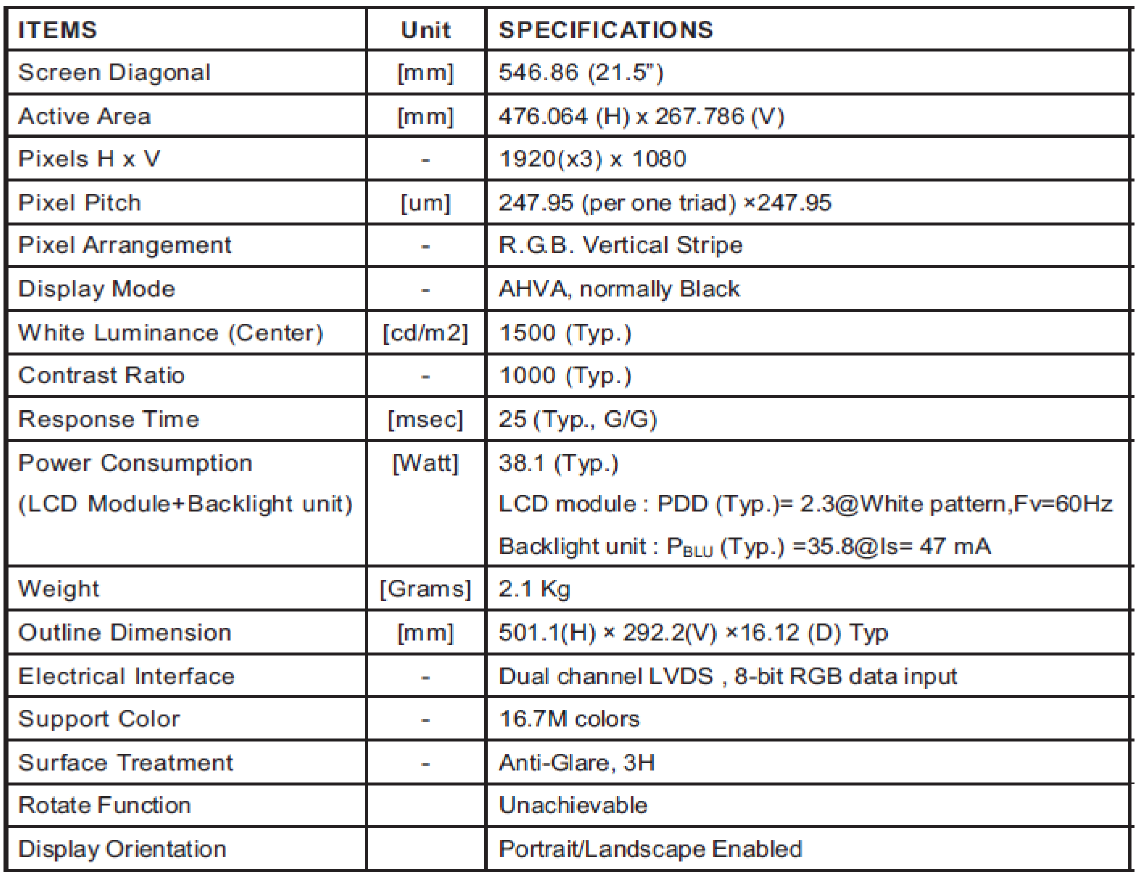
4.8.2 FEATURES OF LIQUID CRYSTAL DISPLAY

- The operating voltage of this LCD is 4.7V-5.3V
- It includes two rows where each row can produce 16-characters.
- The utilization of current is 1mA with no backlight.
- Every character can be built with a 5×8 pixel box.
- The alphanumeric LCDs alphabets & numbers.
- Is display can work on two modes like 4-bit & 8-bit.
- These are obtainable in Blue & Green Backlight.
- It displays a few custom generated characters.

These 16 x 2 LCD display modules are constant of 16 Columns and 2 Rows. The 1st row of this module has a total of 16 columns 0 to 15 and the position of the first row is 0. Also, the 2nd row has a total of 16 columns 0 to 15 and the position of the second row is position is 1. So, the total numbers of the column are 16 x 2 = 32. Its means 16 x 2 LCD module can display 32 characters at the same time.

The LCD consists of Data, Command and control registers. All the register helps to control the different kinds of functions on the LCD. The data and command registers take the input from digital pins D0-D7. Then controls pins help to differentiate between command/data registers.

4.8.3 SPECIFICATIONS OF LIQUID CRYSTAL DISPLAY



ITEMS	Unit	SPECIFICATIONS
Screen Diagonal	[mm]	546.86 (21.5")
Active Area	[mm]	476.064 (H) x 267.786 (V)
Pixels H x V	-	1920(x3) x 1080
Pixel Pitch	[um]	247.95 (per one triad) ×247.95
Pixel Arrangement	-	R.G.B. Vertical Stripe
Display Mode	-	AHVA, normally Black
White Luminance (Center)	[cd/m2]	1500 (Typ.)
Contrast Ratio	-	1000 (Typ.)
Response Time	[msec]	25 (Typ., G/G)
Power Consumption (LCD Module+Backlight unit)	[Watt]	38.1 (Typ.) LCD module : PDD (Typ.)= 2.3@White pattern,Fv=60Hz Backlight unit : P_{BLU} (Typ.) =35.8@Is= 47 mA
Weight	[Grams]	2.1 Kg
Outline Dimension	[mm]	501.1(H) × 292.2(V) ×16.12 (D) Typ
Electrical Interface	-	Dual channel LVDS , 8-bit RGB data input
Support Color	-	16.7M colors
Surface Treatment	-	Anti-Glare, 3H
Rotate Function		Unachievable
Display Orientation		Portrait/Landscape Enabled

Table-4.2: Specifications of LCD

4.8.4 APPLICATIONS OF LCD

- In most of the applications that's have only small values to show, uses the LCD.
- Most of the commercial meters use this module to represent the data output.

- In the toys and developing projects, it is still vastly in use.
- Black and white printers, it helps to show the printer settings and status.

4.9 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.

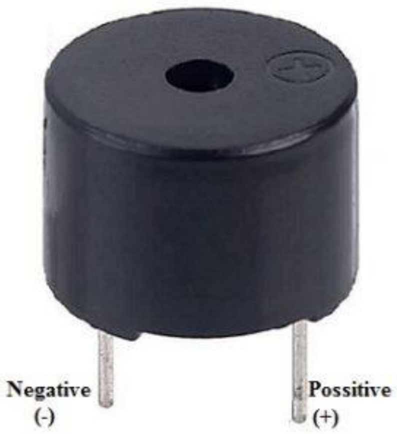


Fig 4.10: Buzzer

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.

4.9.1 SPECIFICATIONS AND FEATURES OF BUZZER

- Rated Voltage: 6V DC
- Operating Voltage: 4-8V DC
- Rated current: <30mA

- Sound Type: Continuous Beep
- Resonant Frequency: ~2300 Hz
- Small and neat sealed package
- Breadboard and Perf board friendly

4.10 POWER SUPPLY

4.10.1 INTRODUCTION

There are many types of power supply. Most are designed to convert high voltage AC mains electricity to a suitable low voltage supply for electronic circuits and other devices. A power supply can by broken down into a series of blocks, each of which performs a particular function. For example, the **Arduino Adapter** is a12V regulated supply can be shown as below

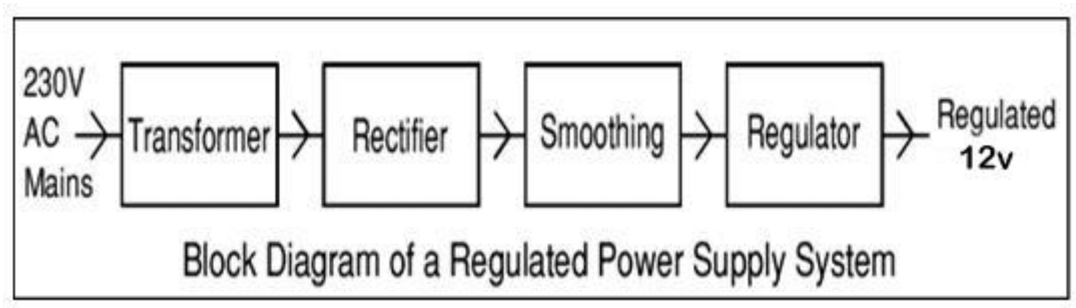


Fig 4.11: Block Diagram of a Regulated Power Supply System

Similarly, 15v regulated supply can also be produced by suitable selection of the individual elements. Each of the blocks is described in detail below and the power supplies made from these blocks are described below with a circuit diagram and a graph of their output:

4.10.2 TRANSFORMER

A transformer steps down high voltage AC mains to low voltage AC. Here we are using a center-tap transformer whose output will be sinusoidal with 12 volts peak to peak value.

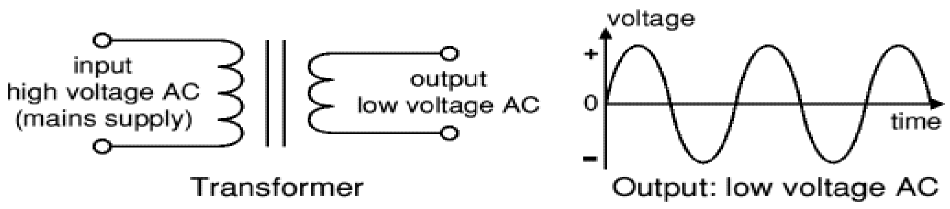


Fig 4.12: Output Waveform of transformer

The low voltage AC output is suitable for lamps, heaters and special AC motors. It is not suitable for electronic circuits unless they include a rectifier and a smoothing capacitor. The transformer output is given to the rectifier circuit.

4.10.3 RECTIFIER

A rectifier converts AC to DC, but the DC output is varying. There are several types of rectifiers; here we use a bridge rectifier.

The Bridge rectifier is a circuit, which converts an ac voltage to dc voltage using both half cycles of the input ac voltage. The Bridge rectifier circuit is shown in the figure. The circuit has four diodes connected to form a bridge. The ac input voltage is applied to the diagonally opposite ends of the bridge. The load resistance is connected between the other two ends of the bridge.

For the positive half cycle of the input ac voltage, diodes D1 and D3 conduct, whereas diodes D2 and D4 remain in the OFF state. The conducting diodes will be in series with the load resistance R_L and hence the load current flows through $R_{L.}$

For the negative half cycle of the input ac voltage, diodes D2 and D4 conduct whereas, D1 and D3 remain OFF. The conducting diodes D2 and D4 will be in series with the load resistance R_L and hence the current flows through R_L in the same direction as in the previous half cycle. Thus, a bi-directional wave is converted into unidirectional.

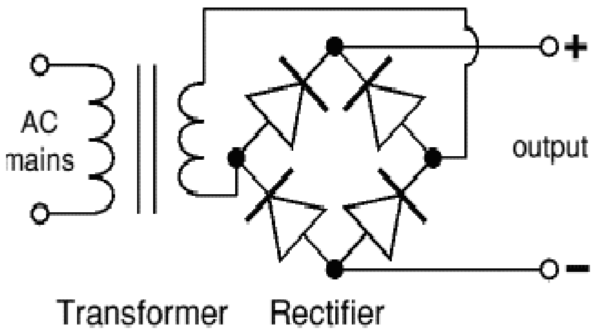


Fig 4.13: Rectifier circuit

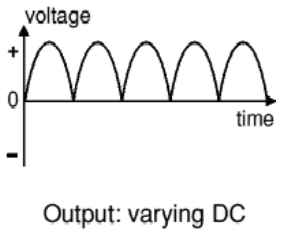


Fig 3.14: Output of the Rectifier

The varying DC output is suitable for lamps, heaters and standard motors. It is not suitable for lamps, heaters and standard motors. It is not suitable for electronic circuits unless they include a smoothing capacitor.

4.10.4 SMOOTHING

The smoothing block smoothes the DC from varying greatly to a small ripple and the ripple voltage is defined as the deviation of the load voltage from its DC value. Smoothing is also named as filtering.

Filtering is frequently affected by shunting the load with a capacitor. The action of this system depends on the fact that the capacitor stores energy during the conduction period and delivers this energy to the loads during the no conducting period. In this way, the time during which the current passes through the load is prolonging Ted, and the ripple is considerably decreased. The action of the capacitor is shown with the help of waveform.

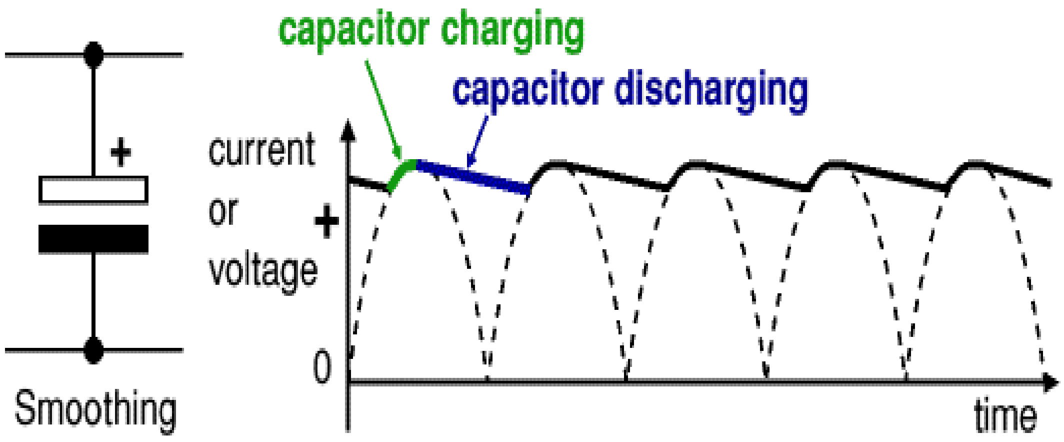


Fig 4.15: Smoothing action of capacitor

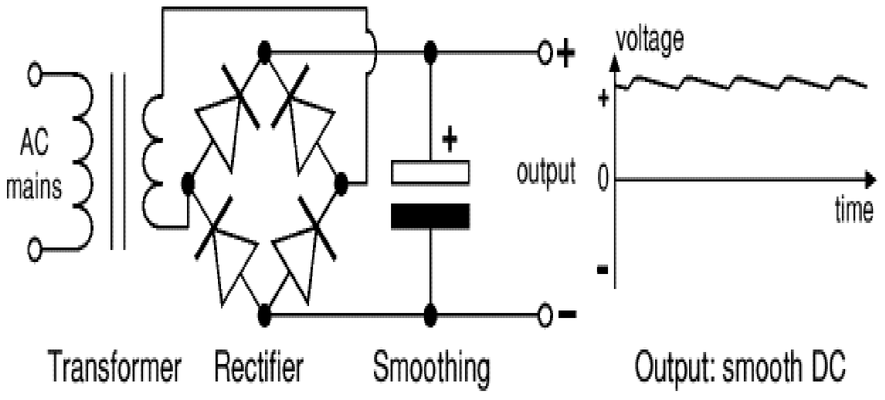


Fig 4.16: Waveform of the rectified output smoothing

4.10.5 REGULATOR

Regulator eliminates ripple by setting DC output to a fixed voltage. Voltage regulator ICs are available with fixed (typically 5V, 12V and 15V) or variable output voltages. Negative voltage regulators are also available. Many of the fixed voltage regulator ICs has 3 leads (input, output and high impedance). They include a hole for attaching a heat sink if necessary. Zener diode is an example of fixed regulator which is shown here.

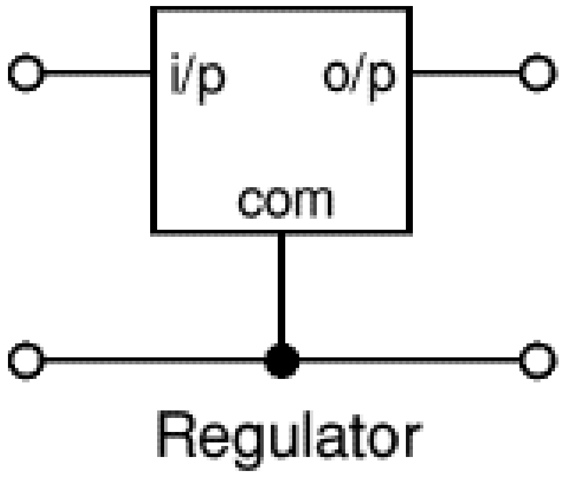


Fig 4.17: Regulator

Transformer + Rectifier + Smoothing + Regulator:

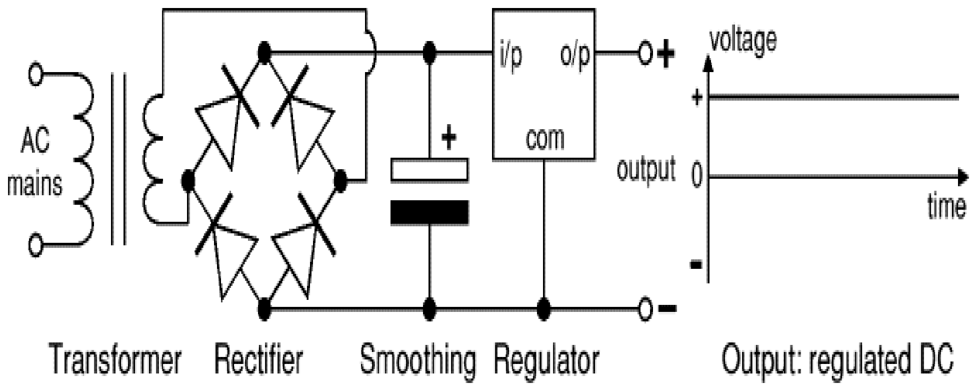


Fig 4.18: Full wave bridge rectifier

CHAPTER-5
SOFTWARE AND CODE

5.1 ARDUINO IDE

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

5.1.1 About the Arduino IDE Tools

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board. In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1 – First you must have your Arduino board (you can choose your favourite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

Fig 5.1: A to B standard USB cable

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

Step 2 – Download Arduino IDE Software.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.

Step 3 – Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (33abelled PWR) should glow.

Step 4 – Launch Arduino IDE.

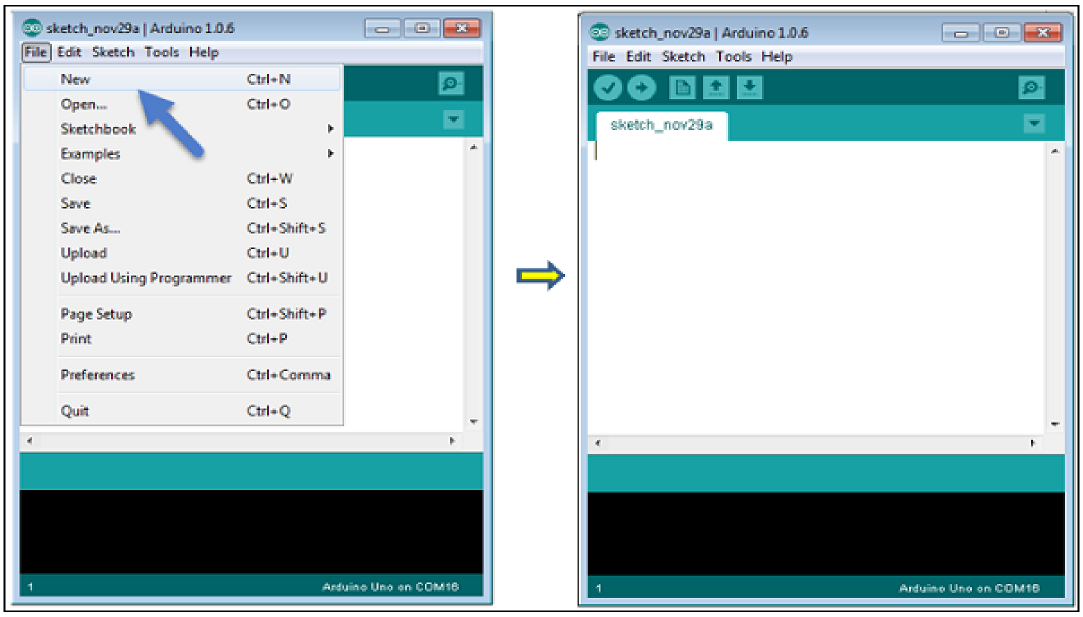
After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.

Step 5 – Open your first project.

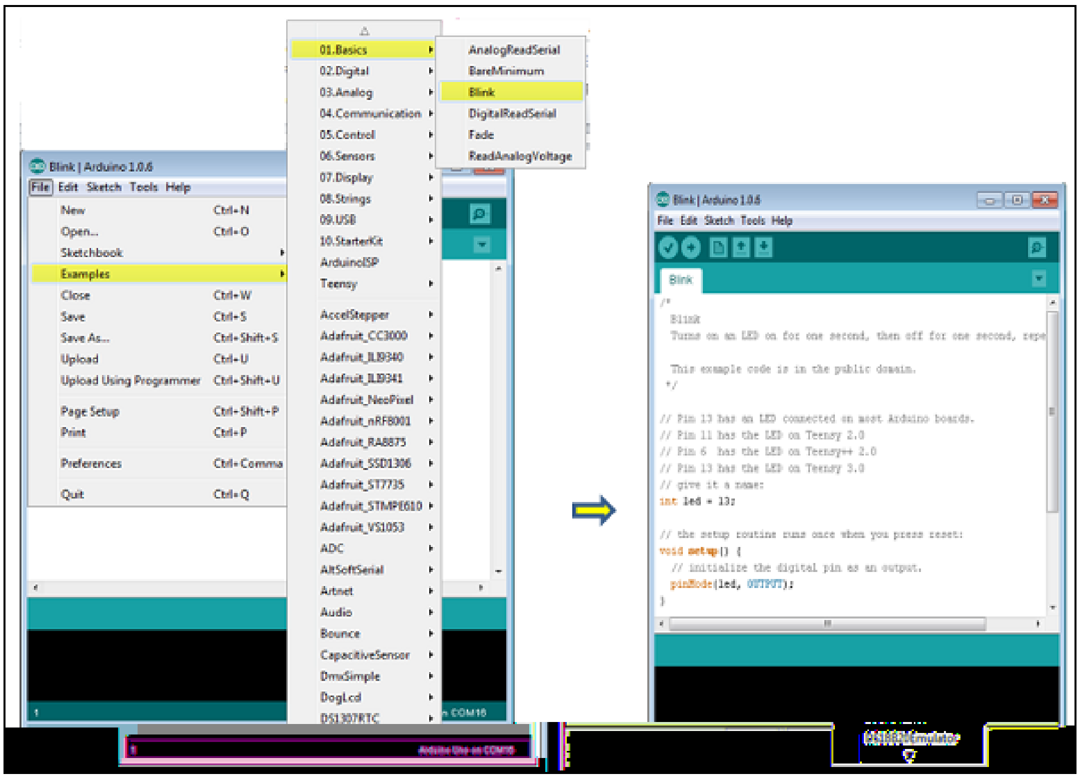
Once the software starts, you have two options –

- Create a new project.
- Open an existing project example.

To create a new project, select File → **New**.



To open an existing project example, select File → Example → Basics → Blink.



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6  has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

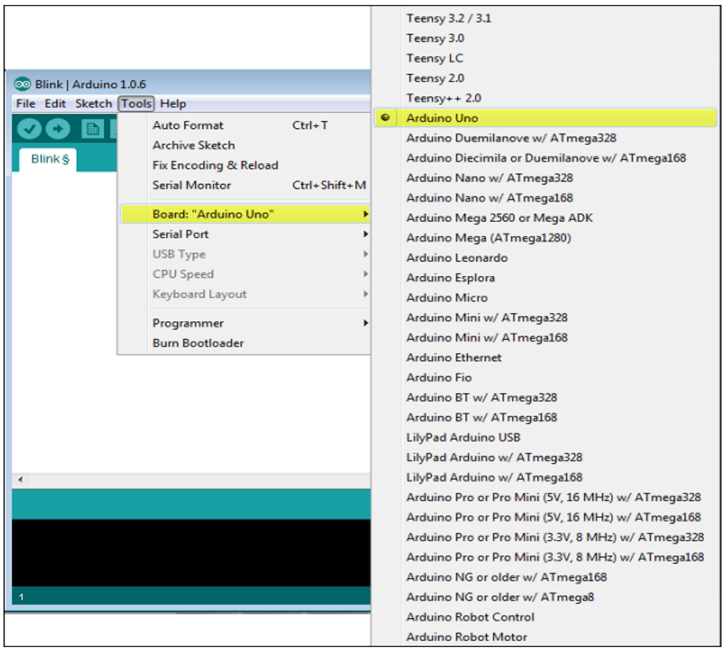
Step 6 – Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

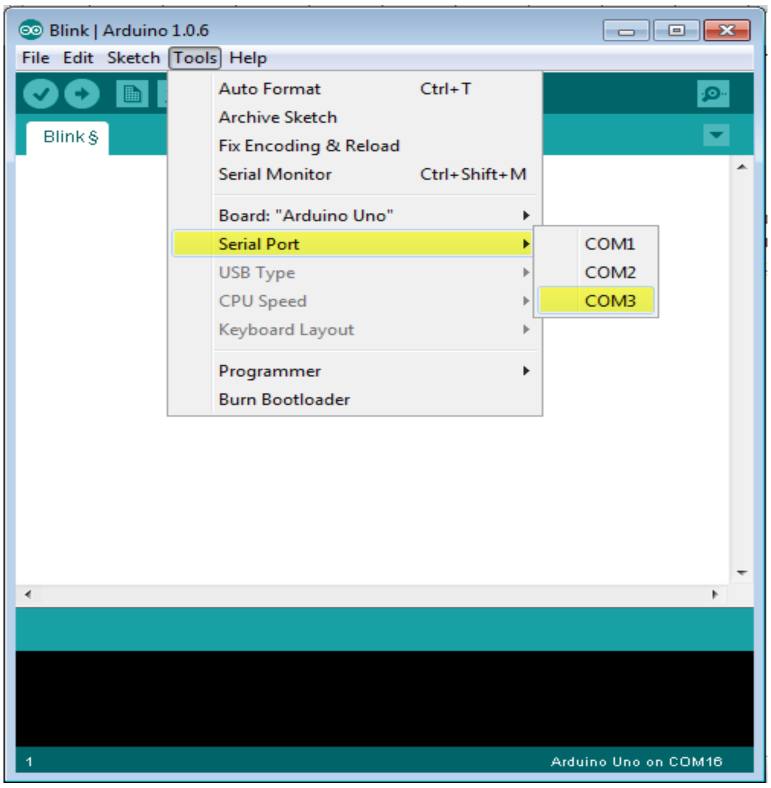
Go to Tools → Board and select your board.

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using.

Step 7 – Select your serial port.

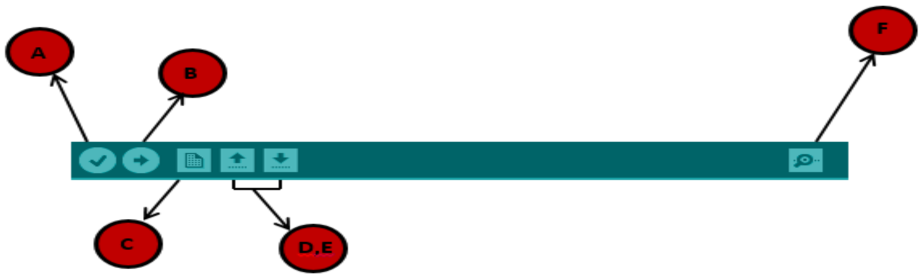

Blink | Arduino 1.0.6
File Edit Sketch Tools Help
Blink §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Arduino Uno"
Serial Port
USB Type
CPU Speed
Keyboard Layout
Programmer
Burn Bootloader
Teensy 3.2 / 3.1
Teensy 3.0
Teensy LC
Teensy 2.0
Teensy++ 2.0
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Robot Control
Arduino Robot Motor
1

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



Step 8 – Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A – Used to check if there is any compilation error.

B – Used to upload a program to the Arduino board.

C – Shortcut used to create a new sketch.

D – Used to directly open one of the example sketches.

E – Used to save your sketch.

F – Serial monitor used to receive serial data from the board and send the serial data to the board.

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

Note – If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

5.2 SOFTWARE CODE

```
int adc_value = 0;
 #include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS1 5
OneWire oneWire1(ONE_WIRE_BUS1);
DallasTemperature sensors1(&oneWire1);

#include <LiquidCrystal.h>
const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
String number="9014475682";
int buzzer=A0;

#include <MAX3010x.h>
#include "filters.h"

// Sensor (adjust to your sensor type)
MAX30105 sensor;
const auto kSamplingRate = sensor.SAMPLING_RATE_400SPS;
const float kSamplingFrequency = 400.0;

// Finger Detection Threshold and Cooldown
const unsigned long kFingerThreshold = 10000;
const unsigned int kFingerCooldownMs = 500;

// Edge Detection Threshold (decrease for MAX30100)
const float kEdgeThreshold = -2000.0;

// Filters
const float kLowPassCutoff = 5.0;
const float kHighPassCutoff = 0.5;

// Averaging
const bool kEnableAveraging = false;
const int kAveragingSamples = 5;
const int kSampleThreshold = 5;


#include<SoftwareSerial.h>
SoftwareSerial gps(6,7);
int i=0,k=0;
int  gps_status=0;
float latitude=17.43601;
float logitude=78.44089;
//float latitude=0;
//float logitude=0;
String Speed="";
String gpsString="";
char *test="$GPRMC";
```

```
String location=" http://maps.google.com/maps?&z=15&mrt=yp&t=k&q="+String(latitude,6)+" +"+String(logitude,6);



void setup()
{
  pinMode(buzzer,OUTPUT);digitalWrite(buzzer,HIGH);
  lcd.begin(16, 2);
  lcd.clear();lcd.print("PATIENT HEALTH");
  lcd.setCursor(0,1);lcd.print("MONI SYSTEM");delay(1000);
  Serial.begin(9600);delay(1000);
lcd.clear();lcd.print("AT");Serial.print("AT\r\n");delay(1000);
lcd.clear();lcd.print("ATE0");Serial.print("ATE0\r\n");delay(1000);
lcd.clear();lcd.print("AT+CMGF=1");Serial.print("AT+CMGF=1\r\n");delay(1000);
lcd.clear();lcd.print("AT+CNMI=1,2,0,0");Serial.print("AT+CNMI=1,2,0,0\r\n");delay(1000);
Serial.print("AT+CMGS=");
Serial.print("");
Serial.print(number);
Serial.print("");
Serial.print("\r\n");delay(1000);
Serial.print(number);Serial.print(":Number Registed");delay(100);
Serial.write(0x1A);delay(10000);

 gps.begin(9600);
  lcd.clear();lcd.print("GPS is Ready");delay(1000);
  lcd.clear(); lcd.print("System Ready");

  if(sensor.begin() && sensor.setSamplingRate(kSamplingRate)) {
    //Serial.println("Sensor initialized");
  }
  else {
    //Serial.println("Sensor not found");
    while(1);
  }
}

// Filter Instances
LowPassFilter low_pass_filter_red(kLowPassCutoff, kSamplingFrequency);
LowPassFilter low_pass_filter_ir(kLowPassCutoff, kSamplingFrequency);
HighPassFilter high_pass_filter(kHighPassCutoff, kSamplingFrequency);
Differentiator differentiator(kSamplingFrequency);
MovingAverageFilter<kAveragingSamples> averager_bpm;
MovingAverageFilter<kAveragingSamples> averager_r;
MovingAverageFilter<kAveragingSamples> averager_spo2;

// Statistic for pulse oximetry
MinMaxAvgStatistic stat_red;
MinMaxAvgStatistic stat_ir;

// R value to SpO2 calibration factors
// See https://www.maximintegrated.com/en/design/technical-documents/app-
notes/6/6845.html
float kSpO2_A = 1.5958422;
float kSpO2_B = -34.6596622;
float kSpO2_C = 112.6898759;

// Timestamp of the last heartbeat
long last_heartbeat = 0;

// Timestamp for finger detection
long finger_timestamp = 0;
```

```
bool finger_detected = false;

// Last diff to detect zero crossing
float last_diff = NAN;
bool crossed = false;
long crossed_time = 0;

void loop() {
  auto sample = sensor.readSample(1000);
  float current_value_red = sample.red;
  float current_value_ir = sample.ir;

  // Detect Finger using raw sensor value
  if(sample.red > kFingerThreshold) {
    if(millis() - finger_timestamp > kFingerCooldownMs) {
      finger_detected = true;
    }
  }
  else {
    // Reset values if the finger is removed
    differentiator.reset();
    averager_bpm.reset();
    averager_r.reset();
    averager_spo2.reset();
    low_pass_filter_red.reset();
    low_pass_filter_ir.reset();
    high_pass_filter.reset();
    stat_red.reset();
    stat_ir.reset();

    finger_detected = false;
    finger_timestamp = millis();
```

```
}

if(finger_detected) {
  current_value_red = low_pass_filter_red.process(current_value_red);
  current_value_ir = low_pass_filter_ir.process(current_value_ir);

  // Statistics for pulse oximetry
  stat_red.process(current_value_red);
  stat_ir.process(current_value_ir);

  // Heart beat detection using value for red LED
  float current_value = high_pass_filter.process(current_value_red);
  float current_diff = differentiator.process(current_value);

  // Valid values?
  if(!isnan(current_diff) && !isnan(last_diff)) {

    // Detect Heartbeat - Zero-Crossing
    if(last_diff > 0 && current_diff < 0) {
      crossed = true;
      crossed_time = millis();
    }

    if(current_diff > 0) {
      crossed = false;
    }

    // Detect Heartbeat - Falling Edge Threshold
    if(crossed && current_diff < kEdgeThreshold) {
      if(last_heartbeat != 0 && crossed_time - last_heartbeat > 300) {
        // Show Results
        int bpm = 60000/(crossed_time - last_heartbeat);
        float rred = (stat_red.maximum()-stat_red.minimum())/stat_red.average();
        float rir = (stat_ir.maximum()-stat_ir.minimum())/stat_ir.average();
        float r = rred/rir;
        float spo2 = kSpO2_A * r * r + kSpO2_B * r + kSpO2_C;

        if(bpm > 50 && bpm < 250) {
          // Average?
          if(kEnableAveraging) {
            int average_bpm = averager_bpm.process(bpm);
            int average_r = averager_r.process(r);
            int average_spo2 = averager_spo2.process(spo2);

            // Show if enough samples have been collected
            if(averager_bpm.count() >= kSampleThreshold) {
              //Serial.print("Time (ms): ");
              //Serial.println(millis());
              //Serial.print("Heart Rate (avg, bpm): ");
              //Serial.println(average_bpm);
              //Serial.print("R-Value (avg): ");
              //Serial.println(average_r);
              //Serial.print("SpO2 (avg, %): ");
              //Serial.println(average_spo2);
            lcd.clear();lcd.print("SpO2 (avg, %): "); lcd.print(average_spo2);delay(10);
            lcd.setCursor(0,1);Serial.print("Heart Rate:");lcd.print(average_bpm);delay(1000);
                sensors1.requestTemperatures();delay(10);
  float ds18b20 = sensors1.getTempCByIndex(0);
lcd.clear();lcd.print("Temp:");lcd.print((float)ds18b20);lcd.print("*C");delay(2000);
int x=analogRead(A2);delay(10);
int y=analogRead(A3);delay(10);

lcd.clear();lcd.print("x:");lcd.print(x);lcd.print(" y:");lcd.print(y);delay(1000);
if(x>380 || x<300 || y>380 || y<300)
{
digitalWrite(buzzer,LOW);delay(1000);digitalWrite(buzzer,HIGH);delay(100
0);
lcd.clear();lcd.print("TILT ALERT");delay(1000);
lcd.clear();lcd.print("AT");Serial.print("AT\r\n");delay(1000);
lcd.clear();lcd.print("ATE0");Serial.print("ATE0\r\n");delay(1000);
lcd.clear();lcd.print("AT+CMGF=1");Serial.print("AT+CMGF=1\r\n");delay(1
000);
lcd.clear();lcd.print("AT+CNMI=1,2,0,0");Serial.print("AT+CNMI=1,2,0,0\r\
n");delay(1000);
Serial.print("AT+CMGS=");
Serial.print("");
Serial.print(number);
Serial.print("");
Serial.print("\r\n");delay(1000);
Serial.print("TILT ALERT");Serial.println(location);delay(100);
Serial.write(0x1A);delay(10000);
String iot="TILT ALERT"+location;delay(1000);
gps.println(iot);delay(1000);
}

if(ds18b20>35)
{
  lcd.clear();lcd.print("HIGH TEMP ALERT");delay(1000);

digitalWrite(buzzer,LOW);delay(1000);digitalWrite(buzzer,HIGH);delay(100
0);
  lcd.clear();lcd.print("AT");Serial.print("AT\r\n");delay(1000);
lcd.clear();lcd.print("ATE0");Serial.print("ATE0\r\n");delay(1000);
lcd.clear();lcd.print("AT+CMGF=1");Serial.print("AT+CMGF=1\r\n");delay(1
000);
lcd.clear();lcd.print("AT+CNMI=1,2,0,0");Serial.print("AT+CNMI=1,2,0,0\r\
n");delay(1000);
Serial.print("AT+CMGS=");
Serial.print("");
Serial.print(number);
Serial.print("");
Serial.print("\r\n");delay(1000);
Serial.print("HIGH                                    TEMPERATURE
ALERT");Serial.println(location);delay(100);
Serial.write(0x1A);delay(10000);
String iot="HIGH TEMPERATURE ALERT"+location;delay(1000);
gps.println(iot);delay(1000);



}


        }
       }
       else {
        //Serial.print("Time (ms): ");
        //Serial.println(millis());
        lcd.clear();lcd.print("Heart Rate:");lcd.print(bpm);delay(10);
        lcd.setCursor(0,1);lcd.print("SpO2:");lcd.print(spo2);delay(1000);
          sensors1.requestTemperatures();delay(10);
  float ds18b20 = sensors1.getTempCByIndex(0);
lcd.clear();lcd.print("Temp:");lcd.print((float)ds18b20);lcd.print("*C");delay(2
000);
int x=analogRead(A2);delay(10);
int y=analogRead(A3);delay(10);
```

```
lcd.clear();lcd.print("x:");lcd.print(x);lcd.print(" y:");lcd.print(y);delay(1000);
if(x>380 || x<300 || y>380 || y<300)
{
digitalWrite(buzzer,LOW);delay(1000);digitalWrite(buzzer,HIGH);delay(100
0);
lcd.clear();lcd.print("TILT ALERT");delay(1000);
lcd.clear();lcd.print("AT");Serial.print("AT\r\n");delay(1000);
lcd.clear();lcd.print("ATE0");Serial.print("ATE0\r\n");delay(1000);
lcd.clear();lcd.print("AT+CMGF=1");Serial.print("AT+CMGF=1\r\n");delay(1
000);
lcd.clear();lcd.print("AT+CNMI=1,2,0,0");Serial.print("AT+CNMI=1,2,0,0\r\
n");delay(1000);
Serial.print("AT+CMGS=");
Serial.print("");
Serial.print(number);
Serial.print("");
Serial.print("\r\n");delay(1000);
Serial.print("TILT ALERT");Serial.println(location);delay(100);
Serial.write(0x1A);delay(10000);
String iot="TILT ALERT"+location;delay(1000);
gps.println(iot);delay(1000);

}
if(ds18b20>35)
{
  lcd.clear();lcd.print("HIGH TEMP ALERT");delay(1000);

digitalWrite(buzzer,LOW);delay(1000);digitalWrite(buzzer,HIGH);delay(100
0);
  lcd.clear();lcd.print("AT");Serial.print("AT\r\n");delay(1000);
lcd.clear();lcd.print("ATE0");Serial.print("ATE0\r\n");delay(1000);
lcd.clear();lcd.print("AT+CMGF=1");Serial.print("AT+CMGF=1\r\n");delay(1
000);
lcd.clear();lcd.print("AT+CNMI=1,2,0,0");Serial.print("AT+CNMI=1,2,0,0\r\
n");delay(1000);
Serial.print("AT+CMGS=");
Serial.print("");
Serial.print(number);
Serial.print("");
Serial.print("\r\n");delay(1000);
Serial.print("HIGH                                    TEMPERATURE
ALERT");Serial.println(location);delay(100);
Serial.write(0x1A);delay(10000);
String iot="HIGH TEMPERATURE ALERT"+location;delay(1000);
gps.println(iot);delay(1000);

}
      }
     }

     // Reset statistic
     stat_red.reset();
     stat_ir.reset();
    }

    crossed = false;
    last_heartbeat = crossed_time;
   }
  }

  last_diff = current_diff;
 }
}
```

```
void gpsEvent()
{
  gpsString="";
  while(1)
  {
   while (gps.available()>0)            //Serial incoming data from GPS
   {
    char inChar = (char)gps.read();
     gpsString+= inChar;                //store incoming data from GPS to temparary
string str[]
     i++;
    // Serial.print(inChar);
     if (i < 7)
     {
      if(gpsString[i-1] != test[i-1])        //check for right string
      {
        i=0;
        gpsString="";
      }
     }
    if(inChar=='\r')
    {
     if(i>60)
     {
       gps_status=1;
       break;
     }
     else
     {
       i=0;
     }
    }
  }
   if(gps_status)
    break;
  }
}
void get_gps()
{
  lcd.clear();
  lcd.print("Getting GPS Data");
  lcd.setCursor(0,1);
  lcd.print("Please Wait.....");
   gps_status=0;
   int x=0;
   while(gps_status==0)
   {
    gpsEvent();
    int str_lenth=i;
    //coordinate2dec();
    i=0;x=0;
    str_lenth=0;
   }
}
void gpsdata()
{
    lcd.clear();
    lcd.print("Lat:");
    lcd.print(latitude);
    lcd.setCursor(0,1);
    lcd.print("Log:");
    lcd.print(logitude);
    delay(2000);
    lcd.clear();

  }
  void coordinate2dec()
  {
   String lat_degree="";
    for(i=19;i<=20;i++)
      lat_degree+=gpsString[i];
      Serial.println(lat_degree);
   String lat_minut="";
     for(i=21;i<=30;i++)
      lat_minut+=gpsString[i];
      Serial.println(lat_minut);
   String log_degree="";
    for(i=32;i<=34;i++)
      log_degree+=gpsString[i];
      Serial.println(log_degree);
   String log_minut="";
    for(i=35;i<=44;i++)
      log_minut+=gpsString[i];
      Serial.println(log_minut);

    Speed="";
    for(i=45;i<48;i++)          //extract longitude from string
      Speed+=gpsString[i];

     float minut= lat_minut.toFloat();
     minut=minut/60;
     float degree=lat_degree.toFloat();
     latitude=degree+minut;
     Serial.println(latitude);
     minut= log_minut.toFloat();
     minut=minut/60;
     degree=log_degree.toFloat();
     logitude=degree+minut;
     Serial.println(logitude);
  }

  #include<string.h>
  #include <WiFiClientSecure.h>   // Include the HTTPS library
  #include <ESP8266WiFi.h>        // Include the Wi-Fi library
  #include <ESP8266WiFiMulti.h>   // Include the Wi-Fi-Multi library
  #include "Arduino.h"
  #include <EMailSender.h>
  ESP8266WiFiMulti wifiMulti;             // Create an instance of the
  ESP8266WiFiMulti class, called 'wifiMulti'
  uint8_t connection_state = 0;
  uint16_t reconnect_interval = 10000;
  WiFiClient  client;
  String data1="";
  String data2="cmd";
  String data=" MESSAGE";
  EMailSender emailSend("sivaprasadpilla7@gmail.com", "tvgcjlhvnpdetvvq");
  void gmail()
  {
   EMailSender::EMailMessage message;

    message.subject = "This is an email from hardware system :  "+data1;
    message.message = "This is an email from hardware system : "+data1;

    EMailSender::Response resp =
  emailSend.send("siva9014475682@gmail.com", message);
```

```
    Serial.println("Sending status: ");

    Serial.println(resp.status);
    Serial.println(resp.code);
    Serial.println(resp.desc);
}
void upload()
{
const char* server4 = "api.thingspeak.com";
const char* _getLink4 = "https://api.thingspeak.com/update?api_key=91SHW5GUCCRO6Y5M&field1="; // Thingspeak.com
//const char* _getLink4 = "https://api.thingspeak.com/update?api_key=JO1IQUEG175OGRNB&field1="; // Thingspeak.com

 // Serial.println("data uploading");delay(1000);
  client.connect(server4,80);
 if (client.connect(server4,80))      // "184.106.153.149" or api.thingspeak.com
https://api.thingspeak.com/apps/thinghttp/send_request?api_key=CT9B331KB5PLM1G5
  {
    String getStr4 = _getLink4;
    client.print("GET "+getStr4+data1+"\n");
    client.print("HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n\n\n");
  }
  client.stop();

}
```

```
void readdata()
{
  data1="";delay(1000);
const char* server4 = "api.thingspeak.com";
const                    char*                    _getLink4                    =                    "
https://api.thingspeak.com/channels/562742/fields/1/last.txt";                          //
Thingspeak.com



  //Serial.println("data uploading");delay(1000);
  client.connect(server4,80);
 if (client.connect(server4,80))      // "184.106.153.149" or api.thingspeak.com
https://api.thingspeak.com/apps/thinghttp/send_request?api_key=CT9B331KB
5PLM1G5
  {
    String getStr4 = _getLink4;
    client.print("GET "+getStr4+"\n");
    client.print("HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n\n\n");
    client.available();
    data1=client.readString();delay(1000);
    //Serial.println(data1);delay(1000);

if(data1[0]=='*')
{
  if(data2==data1)
  {


  }
  else
  {
  Serial.println(data1);upload();
  }
  data2=data1;
}
/*



if((data1=="light1on")||(data1=="light1off")||(data1=="light2on")||(data1=="li
ght2off")||(data1=="fan1on")||(data1=="fan1off")||(data1=="fan2on")||(data1=
="fan2off"))
{
  Serial.print(data1);delay(1000);upload();
}

    if(data1[0]=='*')
    {
   Serial.println(data1);delay(10000);upload();
    }
    if((data1=="1")||(data1=="2")||(data1=="3")||(data1=="4")||(data1=="0"))
    {
   Serial.print(data1);delay(10000);
      }
      */
  }
  client.stop();
}




void setup()
{
  Serial.begin(9600);         // Start the Serial communication to send messages to
the computer
  delay(10);
  //Serial.println('\n');

  wifiMulti.addAP("consciencetechnologies", "484conscience777");   // add Wi-
Fi networks you want to connect to
  wifiMulti.addAP("sivaji", "sivaji.123");
  wifiMulti.addAP("ZTE-sUQdqa", "5hjgxyh9");
  wifiMulti.addAP("project", "project.123");
  wifiMulti.addAP("123456789", "123456789");

  //Serial.println("Connecting ...");
  int i = 0;
  while (wifiMulti.run() != WL_CONNECTED) { // Wait for the Wi-Fi to
connect: scan for Wi-Fi networks, and connect to the strongest of the networks
above
    delay(250);
    //Serial.print('.');
  }
  //Serial.println('\n');
  //Serial.print("Connected to ");
  //Serial.println(WiFi.SSID());              // Tell us what network we're connected
to
  //Serial.print("IP address:\t");
 Serial.println(WiFi.localIP());            // Send the IP address of the ESP8266 to
the computer
  //Serial.println('\n');

  //readdata();
//gmail();
}
```

```
void loop()
{

while(1)
{
//readdata();

 while(Serial.available())
 {

  data1=Serial.readString();delay(1000);
   upload();
 /*
if((data1[0]=='P')||(data1[0]=='p')||(data1[0]=='1')||(data1[1]=='p')||(data1[2]=='
P')||(data1[2]=='p')||(data1[3]=='P')||(data1[3]=='p'))
  {
  upload();
  }*/
  //data1.replace("&field2=", " , Voltege= ");delay(1000);
  //data1.replace("&field3=", " , Temperature= ");delay(1000);
  //data1="Current="+data1;delay(1000);
  gmail();

 }

}

}
```

CHAPTER 6

RESULTS

6.1 OFF CONDITION

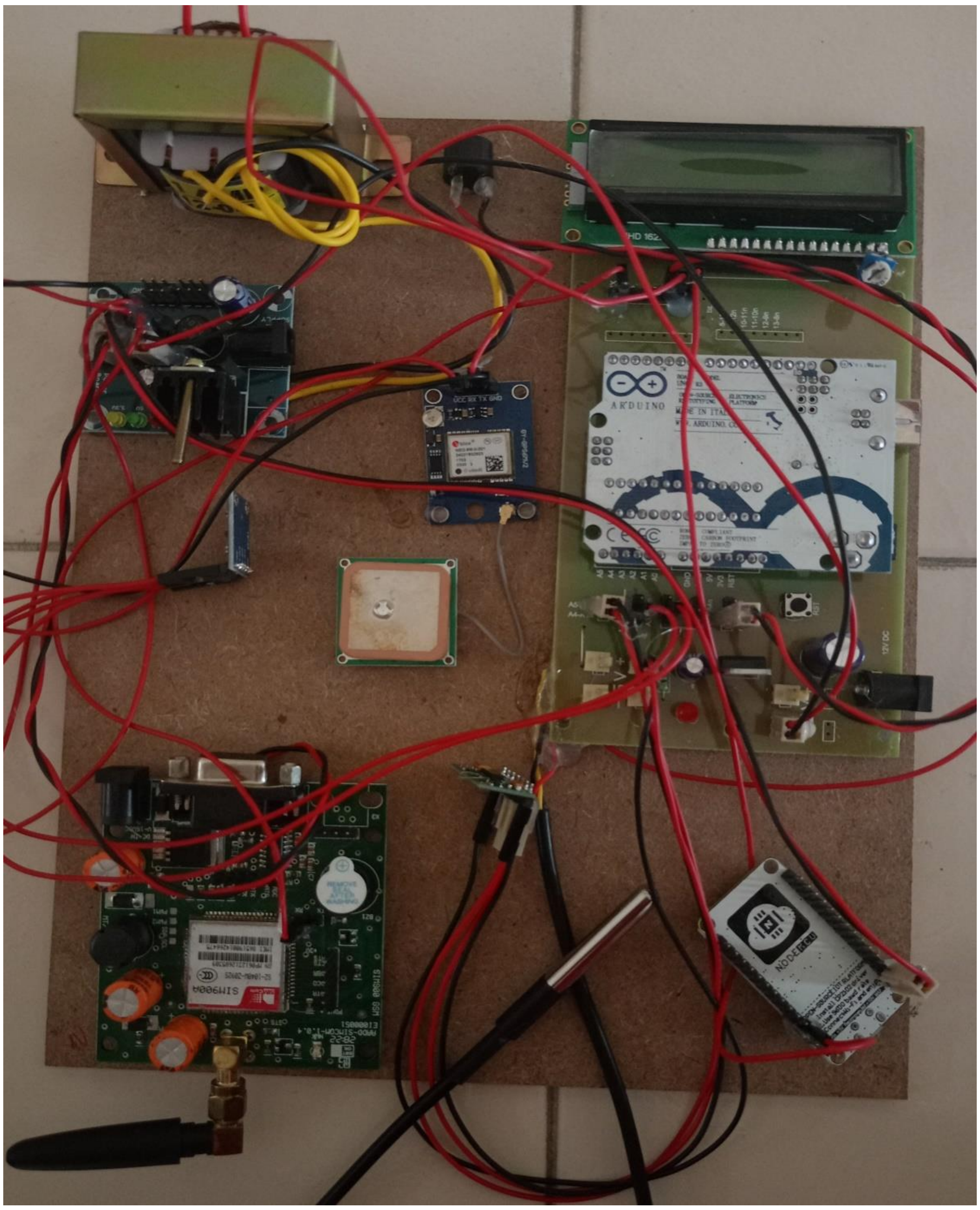


Fig 6.1:OFF Condition

6.2 ON CONDITION

When the kit is in ON condition, the system will look like the fig 6.2 shown below,

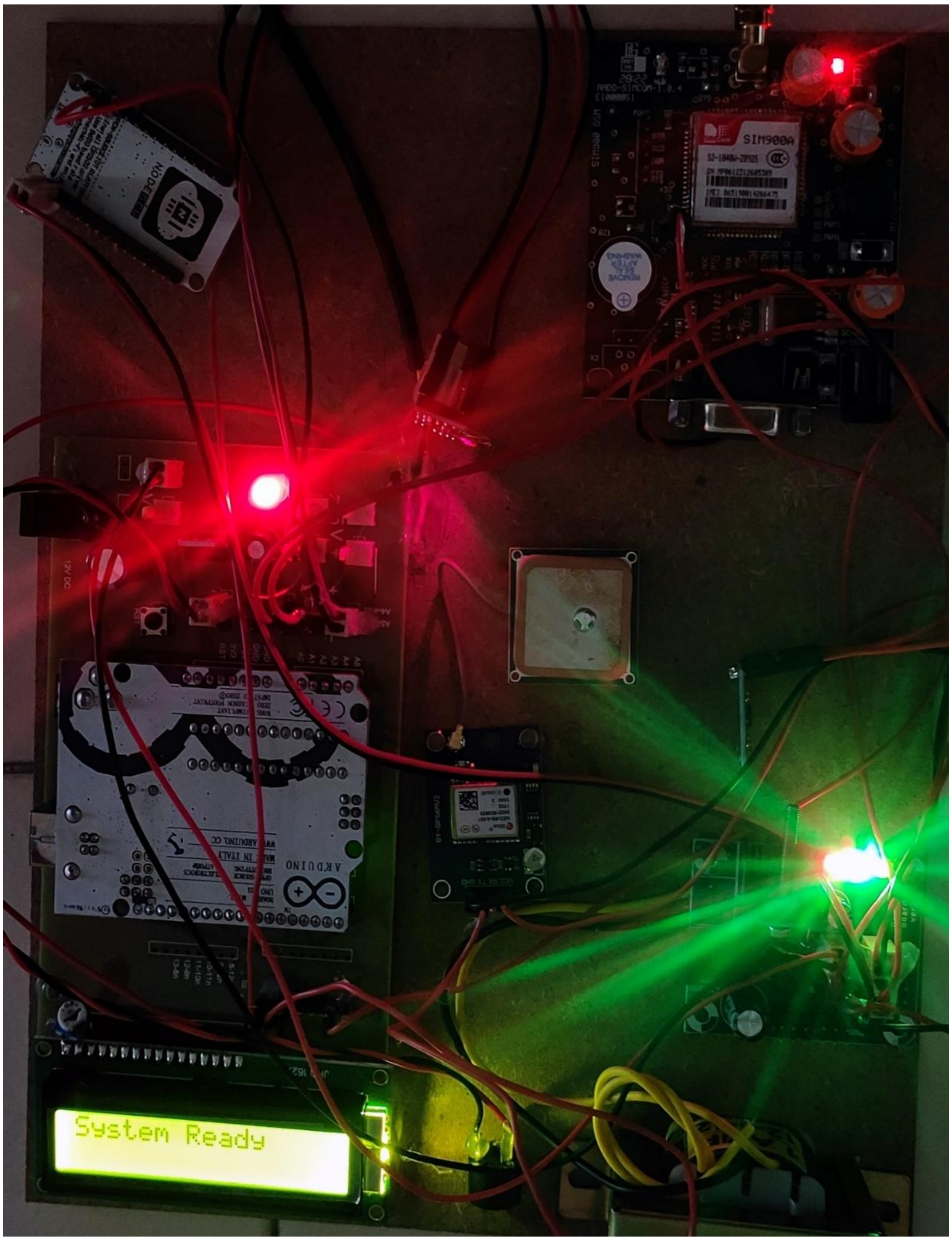
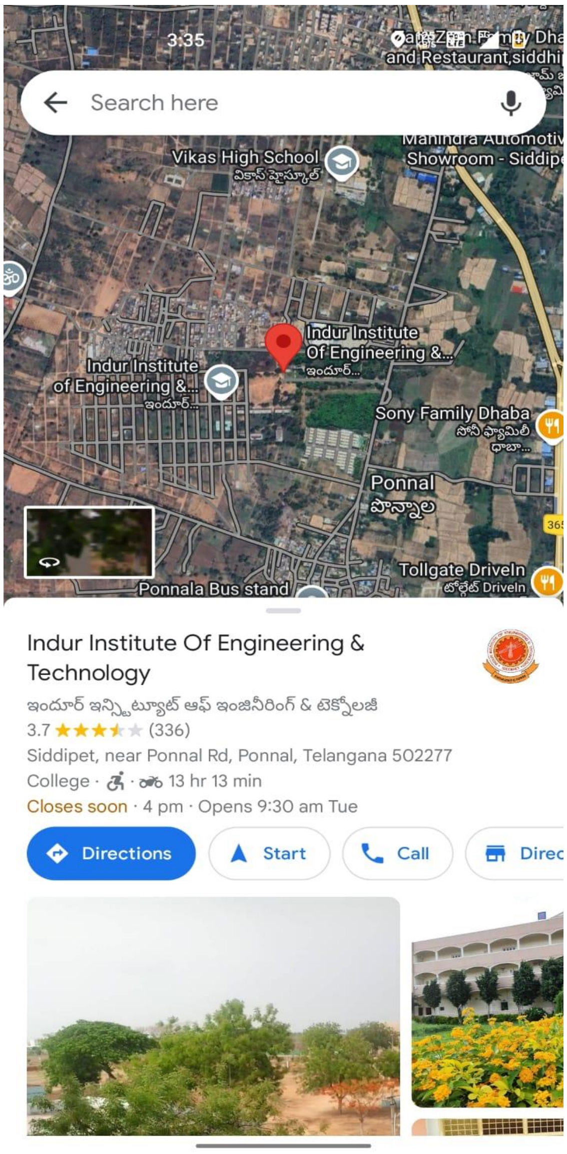


Fig 6.2: ON Condition


3:35
Search here
Vikas High School
Mahindra Automotive Showroom - Siddipet
Indur Institute Of Engineering &...
Indur Institute of Engineering &...
Sony Family Dhaba
Ponnal
Tollgate DriveIn
Ponnala Bus stand
Directions
Start
Call

Indur Institute Of Engineering & Technology

ఇందూర్ ఇన్స్టిట్యూట్ ఆఫ్ ఇంజినీరింగ్ & టెక్నాలజీ

3.7 ★★★★☆ (336)

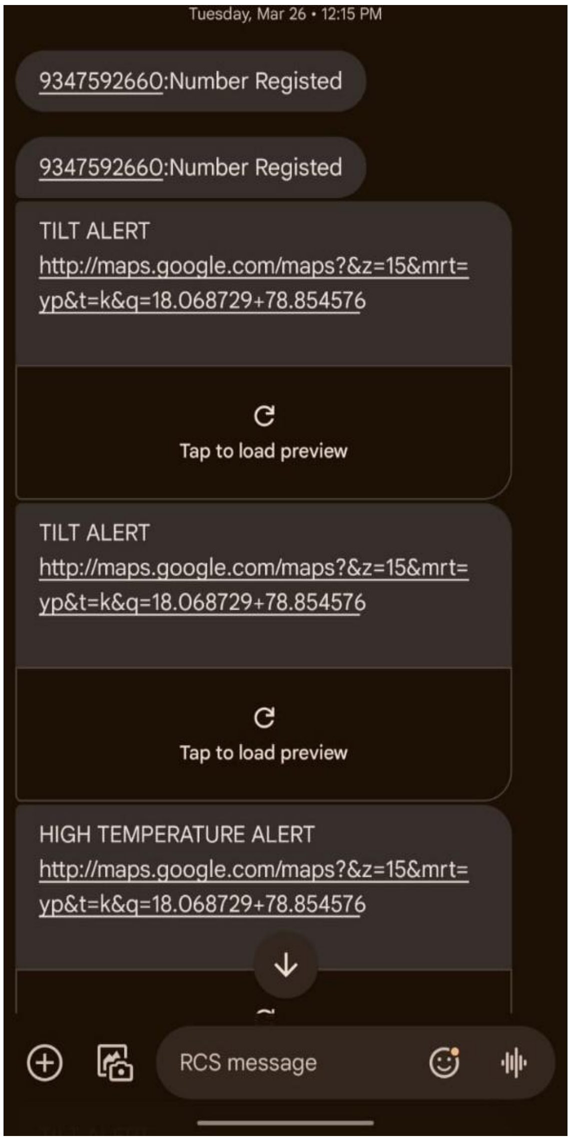
Siddipet, near Ponnal Rd, Ponnal, Telangana 502277

College · 13 hr 13 min

Closes soon · 4 pm · Opens 9:30 am Tue

Fig 6.3: Location

Initially a mobile number will be registered with the GPS Module(Global Positioning System). When the GPS is powered ON, the location of the patient will be sent to the registered mobile number through GSM(Global System for Mobile Communication). The fig 6.3 shown above is the location which was tracked by the GPS and send to the registered mobile number.


Tuesday, Mar 26 • 12:15 PM

9347592660:Number Registed

9347592660:Number Registed

TILT ALERT
http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=18.068729+78.854576

Tap to load preview

TILT ALERT
http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=18.068729+78.854576

Tap to load preview

HIGH TEMPERATURE ALERT
http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=18.068729+78.854576

Fig 6.4: Alert Message

When there is a motion detection, variations in the temperature or Heart rate and Oxygen levels, an alert message will be sent to the registered mobile number.

When the position of TILT Sensor is changed, that means the position of the patient is changed. Then there will be a motion detection and a TILT ALERT message will be sent to the mobile number.

When there are rise or fall in temperature levels or heart rate and oxygen levels, an alert message will be sent through the GSM to the registered number. The fig 6.4 shown above is the messages sent through the GSM to the mobile.

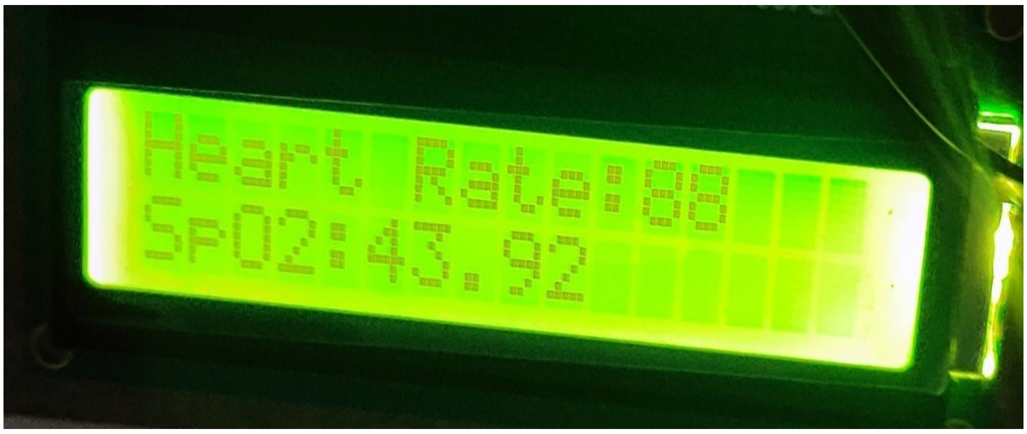


Fig 6.5: Heart Rate and Oxygen Levels

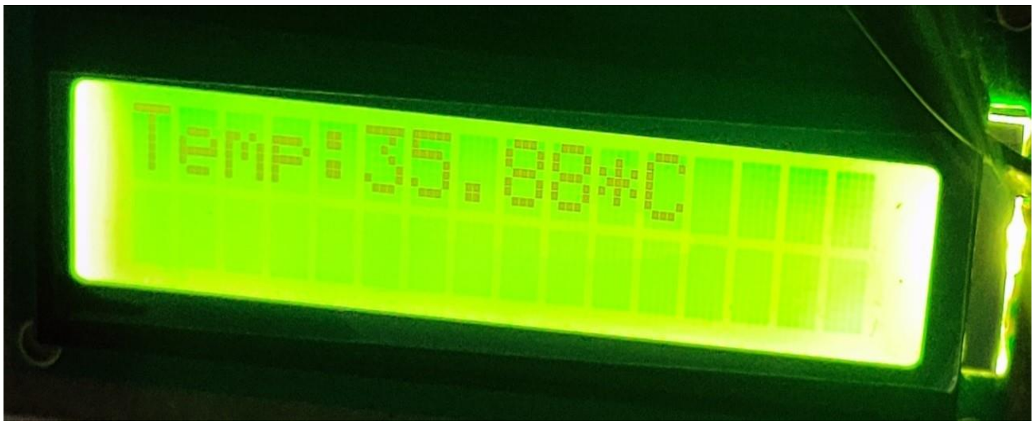


Fig 6.6: Temperature (*C)

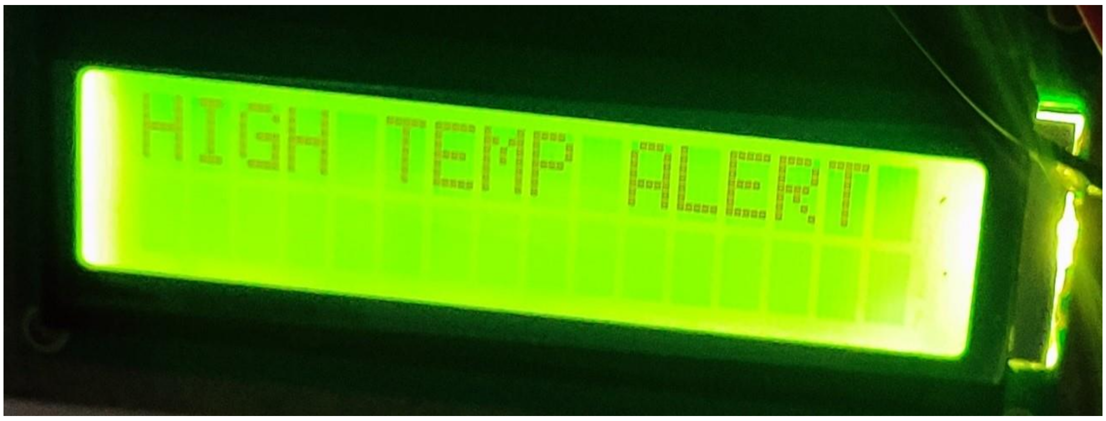


Fig 6.7: High Temperature Alert (*C)

DHT11 sensor will measures the temperature from 0*C to 50*C. The minimum temperature range is 35*C and the maximum temperature range is 37*C. If the temperature goes below 35*C, an alert message will be sent to the mobile showing “LOW TEMP ALERT”. If the temperature exceeds maximum temperature range i.e., 37*C, an alert message will be sent to the mobile showing “HIGH TEMP ALERT”. They are shown above in fig 6.6 and fig 6.7.

6.3 ADVANTAGES

- Efficient treatment
- Reduced risk in critical time
- Connectivity of devices
- Collection and analyze of massive amount of data
- Reduction of health care cost.

6.4 APPLICATIONS

- Clinical trails
- Elderly care
- Emergency response system
- Public health surveillance.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The system introduced is smart healthcare that monitors the patient's basic vital signs like heart rate and temperature. The success rate of observational and actual data is around 95% or higher all told cases of the developed healthcare system. Real medical staff can view and track data in real time, even when the patient is being tested outside the hospital. This method may also analyze raw medical data in an exceedingly short amount of your time, which is additionally beneficial for nurses and doctors in epidemic and crisis situations. The developed prototype is incredibly easy to style and use. The system is extremely useful within the case of communicable disease sort of a novel coronavirus (COVID19) treatment. The developed system will improve this healthcare system which will protect plenty of lives from death. This project is useful for both doctors and patients. Physicians can know the patient's health through IoT-based sensors and can also analyze health data stored in real-time databases. It saves time and the patient gets an instance of the appropriate treatment

7.2 FUTURE SCOPE

The proposed system monitors the patient condition especially for the ICU or cardiac patients but in the future, we will upgrade both hardware and software part. In hardware part, we will measure temperature of the patient so for this we will need temperature sensor. Also, we will monitor the whole ward room or patient room from far places by Wi-Fi module. Therefore, person fall detection feature will be added which would be beneficial to older people.

REFERENCES/ BIBLOGRAPHY

1. B. Rebsamen, C. Guan, H. Zhang, C. Wang, C. Teo, M. H. Ang, Jr., and E. Burdet, “A brain controlled wheelchair to navigate in familiar environments,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 18, no. 6, pp. 590–598, Dec. 2010.
2. J. d. R. Millan, R. Rupp, G. R. Muller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, C. Vidaurre, F. Cincotti, A. Kubler, R. Leeb, C. Neuper, K.- R. Muller, and D. Mattia, “Combining brain– computer interfaces and assistive technologies state-of-the-art and challenges,” Frontiers Neurosci., vol. 4, pp. 1–15, 2010.
3. X. Perrin, “Semi-autonomous navigation of an assistive robot using lo throughput interfaces,” Ph.D. dissertation, ETHZ, Zurich, Switzerland, 2009.
4. Nijholt, D. Tan, G. Pfurtscheller, C. Brunner, J. del R. Millan, B. Allison, B. Graimann, F. Popescu, B. Blankertz, and K.-R. Muller, “Brain–computer interfacing for intelligent systems,” IEEE Intell. Syst., vol. 23, no. 3, pp. 72–79, May/Jun. 2008.
5. J. R. Wolpaw, D. J. McFarland, G. W. Neat, and C. A. Forneris, “An EEGbased brain–computer interface for cursor control,” Electroencephalogr. Clin. Neurophysiol., vol. 78, no. 3, pp. 252–259, Mar. 1991.
    - www.ijsred.com
    - www.microcontrollerstudies/pic16f8xx.html
    - www.touchtech/resistive4wire.html
    - www.engineersgarage.com