ABSTRACT

In the present era, wildlife and forest departments are facing the problem of movement of animals from forest area to residential area. The number of trees has reduced drastically from the forest that creates an unhealthy environment for animals to survive in the forest. It has been found in a survey that 80% losses are caused due to fire. This could have been avoided if the fire was detected in the early stages. This project proposes a system for tracking and alarming for the protection of trees against forest fires

A wildland fire is an uncontrolled fire that occurs mainly in forest areas, although it can also invade urban or agricultural areas. Among the main causes of wildfires, human factors, either intentional or accidental, are the most usual ones. The number and impact of forest fires are expected to grow as a consequence of the global warming. In order to fight against these disasters, it is necessary to adopt a comprehensive, multifaceted approach that enables a continuous situational awareness and instant responsiveness. Thus, by creating a system which can inform us about the potential risk about fire, a necessary action could be taken as quick as possible to eliminate the fire in forest.

INDEX

LIST OF FIGURES

CHAPTER – 1

INTRODUCTION

1.1 Introduction to Embedded System

An embedded system is a special-purpose computer system designed to perform one or few dedicated functions, sometimes with real time computing constraints. It is usually embedded as a part of complete device including hardware and mechanical parts. In contrast general purpose computer, such as 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 cost of the product, or increasing the reliability and performance. Some embedded systems are mass produced, benefiting from economies of scale.

In general ―embedded system‖ is not an exactly defined term, as many systems have some elements 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.

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with a programming interface; an embedded systems programming is a specialized occupation.

The microprocessor-based systems built for controlling a function or range of function's and is not designed to be programmed by the end user in the same way a PC is defined as an embedded system. An embedded system is designed to perform one particular task albeit with different choices and options.

Embedded systems contain processing or that are either microcontrollers or digital signal processors. Microcontrollers generally known as "chip", which may itself be packaged with the

microcontroller hybrid system of Application- Specific Integrated Circuit (ASIC). In general, input always comes from a detector or sensors in more specific word and mean while the output goes to the activator which may start or stop the operation of the machine or the operating system.

An embedded system is a combination of both hardware and software. Each embedded system is unique and the hardware is highly specialized in the application domain. Hardware consists of processors, microcontroller, IR sensors. On the other hand, Software are just like a brain of the whole embedded system as this consists of the programming languages used which make hardware work. As a result, embedded systems programming can be a widely varying experience.

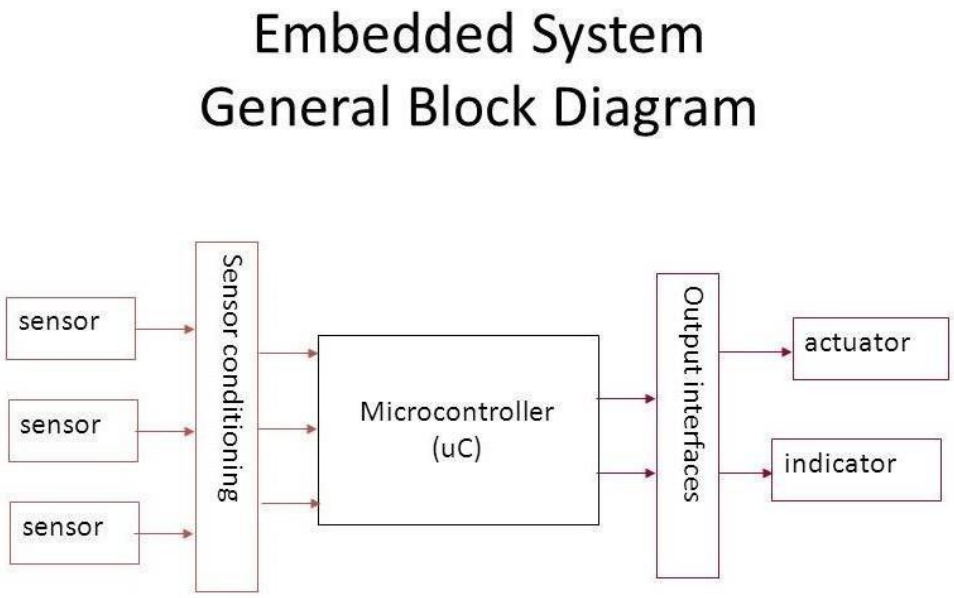


Fig 1: Block diagram for Embedded System

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. 98 percent of all microprocessors are manufactured as components of embedded systems.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them

significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more- complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

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

Embedded systems range from portable devices such as digital watches and MP3 player, to large stationary installations like traffic lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. 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.

Characteristics:

Embedded systems are designed to do some specific task, rather than be a general- purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

Embedded systems are not always standalone devices. Many embedded systems consist of small parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips. They run with limited computer hardware

operating systems. Simple embedded devices use buttons, LEDs, graphic or character LCDs (HD44780 LCD for example) with a simple menu system.

More sophisticated devices which use a graphical screen with touch sensing or screen- edge buttons provide flexibility while minimizing space used: the meaning of the buttons can change with the screen, and selection involves the natural behavior of pointing at what's desired. Handheld systems often have a screen with a "joystick button" for a pointing device.

Some systems provide user interface remotely with the help of a serial (e.g. RS- 232, USB, I²C, etc.) or network (e.g. Ethernet) connection. This approach gives several advantages: extends the capabilities of embedded system, avoids the cost of a display, simplifies BSP and allows one to build a rich user interface on the PC. A good example of this is the combination of an embedded web server running on an embedded device (such as an IP camera) or a network router. The user interface is displayed in a web browser on a PC connected to the device, therefore needing no software to be installed.

1.2 APPLICATIONS OF EMBEDDED SYSTEM

- Military and aerospace software applications
- Communication applications
- Electronic applications and consumer devices
- Industrial automation and process control software

1.3 MICRO PROCESSOR FOR EMBEDDED SYSTEM

Microprocessor is a computer processor that incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits. The microprocessor is a multipurpose, clock driven, register based, programmable electronic device that accepts digital data or binary data as input, processes it according to instructions stored in its memory, and provides results as output. Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary numeral system.

The integration of a whole CPU onto a single chip or on a few chips greatly reduced the cost of processing power. Integrated circuit processors are produced in large numbers by highly automated processes resulting in a low per unit cost. Single-chip processors increase reliability as

there are many fewer electrical connections to fail. As microprocessor designs get faster, the cost of manufacturing a chip (with smaller components built on a semiconductor chip the same size).

Before microprocessors, small computers had been implemented using racks of circuit boards with many medium- and small-scale integrated circuits. Microprocessors integrated this into one or a few large-scale ICs. Continued increases in microprocessor capacity have since rendered other forms of computers almost completely obsolete (see history of computing hardware), with one or more microprocessors used in everything from the smallest embedded systems and handheld devices to the largest mainframes and supercomputers.

A microprocessor is a general-purpose system. Several specialized processing devices have followed from the technology. Microcontrollers integrate a microprocessor with peripheral devices in embedded systems. A digital signal processor (DSP) is specialized for signal processing. Graphics processing units may have no limited or general programming facilities. For example, GPUs through the 1990s were mostly non-programmable and have only recently gained limited facilities like programmable vertex shader.

32-bit processors have more digital logic than narrower processors, so 32-bit (and wider) processors produce more digital noise and have higher static consumption than narrower processors. Reducing digital noise improves ADC conversion results. So, 8-bit or 16-bit processors are better than 32-bit processors for system on a chip and microcontrollers that require extremely low-power electronics, or are part of a mixed-signal integrated circuit with noise-sensitive on-chip analog electronics such as high-resolution Analog to digital converters, or both.

Nevertheless, trade-offs apply: running 32-bit arithmetic on an 8-bit chip could end up using more power, as the chip must execute software with multiple instructions. Modern microprocessors go into low power states when possible, and an 8-bit chip running 32-bit software is active most of the time. This creates a delicate balance between software, hardware and use patterns, plus costs.

When manufactured on a similar process, 8-bit microprocessors use less power when operating and less power when sleeping than 32-bit microprocessors.

However, some people say a 32-bit microprocessor may use less average power than an 8-bit microprocessor when the application requires certain operations such as floating-point math that take many more clock cycles on an 8-bit microprocessor than a 32-bit microprocessor so the 8-bit microprocessor spends more time in high-power operating mode.

Microprocessors are silicon chips that contain a computer's central processing unit

(CPU)—the device that executes commands entered into the computer. Along with clocks and main memory, CPUs are among a computer's main components. The terms CPU and microprocessor often are used interchangeably. Essentially, microprocessors are responsible for manipulating data and performing numeric calculations and logical comparisons. At the heart of microprocessors are tiny electronic switches called transistors, which allow digital computers to process information in the form of electrical signals. These signals are in one of two states (on or off), and are represented by ones and zeroes, respectively. High-level programming languages like Java or C++, used to write popular software programs, eventually are translated to the machine language of ones and zeroes that computers understand.

Intel was the first company to produce a microprocessor for commercial use. Called the 4004, it was released in the early 1970s and contained slightly more than 2,000 transistors. By the early 2000s, microprocessors contained more than 5 million transistors on a single silicon chip. The more transistors a chip has, the more quickly it can process information. A microprocessor's clock speed defines the number of instructions it can carry out per second. This figure is expressed in Megahertz (MHz) or Gigahertz (GHz). In 2001 the processing speeds of some microprocessors exceeded 1.7 GHz.

In 1965, Intel Co-Founder Gordon E. Moore predicted the number of transistors manufacturers could fit onto a silicon chip would double every 18 months. Because his prediction proved to be accurate over time, it came to be known as Moore's Law. The law eventually will expire when it becomes physically impossible for manufacturers to fit any more transistors onto a single chip. This is expected to happen somewhere around 2017 or 2020 when transistors are atom-sized. At that time, a new computing architecture will be necessary. One possibility is quantum computing, which relies on atomic properties instead of transistors to determine the one’s and zero’s a computer understands. According to InfoWorld "quantum computers rely on a particle's traits, such as the direction of its spin, for creating a state. For example, when the spin is up, a particle could be read as 'one,' and when its spin is down, the particle would be read as 'zero.'

In mid-2001 Intel announced experimental technology that it called "Wireless- Internet-On-A-Chip." Essentially, the technology consisted of a silicon chip that held a microprocessor, as well as Analog communication circuits and flash memory. According to Intel, the technology potentially would lead to the development of more powerful wireless Internet devices. Around the same time, Intel and Hewlett-Packard announced the launch of

the Itanium Processor, a new generation of microprocessor the companies co-developed for use in servers and workstation computers.

Microprocessors are the devices in a computer which make things happen. Microprocessors are capable of performing basic arithmetic operations, moving data from place to place, and making basic decisions based on the quantity of certain values.

The vast majority of microprocessors can be found in embedded microcontrollers. The second most common type of processors are common desktop processors, such as Intel's Pentium or AMD's Athlon. Less common are the extremely powerful processors used in high- end servers, such as Sun's SPARC, IBM's Power, or Intel's Itanium.

Historically, microprocessors and microcontrollers have come in "standard sizes" of 8 bits, 16 bits, 32 bits, and 64 bits. These sizes are common, but that does not mean that other sizes are not available. Some microcontrollers (usually specially designed embedded chips) can come in other "non-standard" sizes such as 4 bits, 12 bits, 18 bits, or 24 bits. The number of bits represent how much physical memory can be directly addressed by the CPU. It also represents the number of bits that can be read by one read/write operation. In some circumstances, these are different; for instance, many 8-bit microprocessors have an 8-bit data bus and a 16-bit address bus.

8-bit processors can read/write 1 byte at a time and can directly address 256 bytes

16-bit processors can read/write 2 bytes at a time, and can address 65,536 bytes (64 Kilobytes) 32-bit processors can read/write 4 bytes at a time, and can address 4,294,967,295 bytes (4 Gigabytes). 64-bit processors can read/write 8 bytes at a time, and can address 18,446,744,073,709,551,616 bytes (16 Extra bytes).

Micro Processor:

A microprocessor incorporates the functions of a CPU on a single integrated circuit or a few integrated circuits. It is a computer processor on a microchip and is a multipurpose, programmable device that uses digital data as input and provides results as an output once it processes the input according to instructions stored in its memory. Microprocessors use sequential digital logic as they have internal memory and operate on numbers and symbols represented in the binary numeral system. They are designed to perform arithmetic and logic operations that make use of data on the chip. General purpose microprocessors in PCs are used for multimedia display, computation, text editing and communication. Several microprocessors are part of embedded systems. These embedded microprocessors provide

digital control to several objects including appliances, automobiles, mobile phones and industrial process control.

The microprocessor contains all, or most of, the central processing unit (CPU) functions and is the "engine" that goes into motion when you turn your computer on. A microprocessor is designed to perform arithmetic and logic operations that make use of small number-holding areas called registers. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another. These operations are the result of a set of instructions that are part of the microprocessor design.

When your computer is turned on, the microprocessor gets the first instruction from the basic input/output system (BIOS) that comes with the computer as part of its memory. After that, either the BIOS, or the operating system that BIOS loads into computer memory, or an application program is "driving" the microprocessor, giving it instructions to perform.

Applications of Microprocessor:

- DVD players
- Cellular Telephones
- Household Appliances
- Car Equipment
- Toys
- Light Switches and Dimmers
- Electrical Circuit Breakers
- Smoke Alarms
- Battery Packs
- Car Keys
- Power tool and Test Instruments.

CHAPTER – 2

FOREST FIRE DETECTION SYSTEM

2.1 Introduction:

Forest fire detection system is an embedded system project which involves detection of fire in the forest and then alarming or informing us about the danger by sending an SMS. The process of detection of forest fire initiates at any of the nodes planted on a tree inside the forest. The forest has a network of nodes placed at suitable distances from each other, the nodes have a capability to communicate through devices (RF module in our case) and by using Arduino. If any change above a threshold value is found in the atmospheric parameters (temperature rise, contamination of air with smoke, etc.) near a node (source node), the information is passed to a nearest intermediate node until it reaches to the main/head terminal. The main/head terminal uses a GSM modem to pass the information to a cell phone (the forest fire monitoring center).

2.2 Description:

The forest fire detection module works in three different stages. The first stage consists of reading some external environmental parameters like temperature and smoke. The first stage is done with the help of some sensors which are used to sense and convert analog data to digital data. The sensors read parameters like temperature, humidity and air quality then sends this information to the next nearest node. This process goes on until the information reaches to the final node or the main terminal which is the second stage of the overall process. The third stage consists of transmission of the information to the forest fire monitoring unit. Each node has a temperature and humidity sensor, a smoke sensor and a microcontroller unit. Arduino has been used as the microcontroller device. The sensors interact with the Arduino and store the information for comparison process. There is a predefined threshold value to each of these parameters. The microprocessor compares the sensor values at regular intervals of times with the threshold values. Based on the comparison

if the input values of sensors exceed the threshold the node transmits the information to the next nearby node which again in turn transmits the information to the other nearby node. In this way the message flow is regulated in this model.

2.3 HARDWARE REQUIREMENT:

This chapter briefly explains about the Hardware Implementation of the project. It discusses the design and working of the design with the help of block diagram and circuit diagram and explanation of circuit diagram in detail.

The implementation of the project design can be divided in two parts.

1. Hardware implementation

2. Firmware implementation

Hardware implementation deals in drawing the schematic on the plane paper according to the application, testing the schematic design over the breadboard using the various IC's to find if the design meets the objective, carrying out the PCB layout of the schematic tested on breadboard, finally preparing the board and testing the designed hardware.

The project design and principle are explained in this chapter using the block diagram and circuit diagram. The block diagram discusses about the required components of the design and working condition is explained using circuit diagram and system wiring diagram.

Based on the Processor side Embedded Systems is mainly divided into 3 types:

1. **Micro Processor: -** are for general purpose ex: our personal computer
2. **Micro Controller: -** are for specific applications, because of cheaper cost
3. **DSP (Digital Signal Processor): -** are for high and sensitive application purpose

2.3.1 ARDUINO UNO:

Fig 2: Arduino Uno

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board -- you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

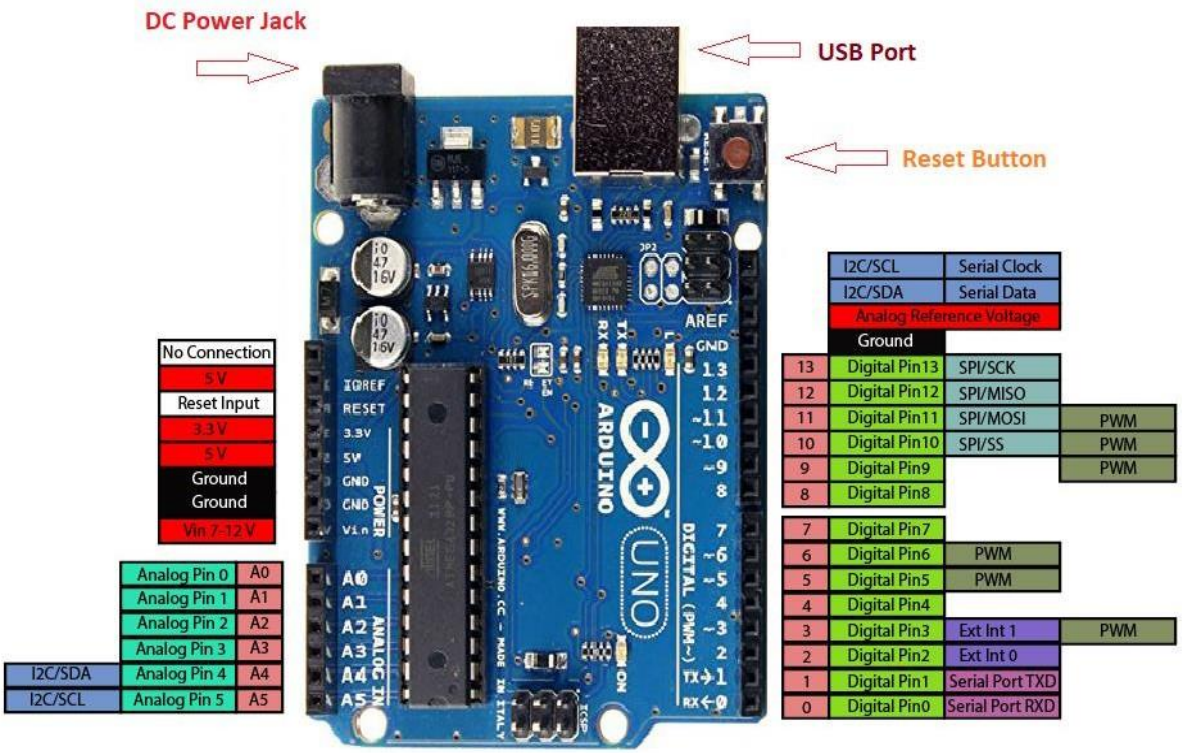


Fig 3: Pin Definition of Arduino Uno

Arduinos contain a number of different parts and interfaces together on a single circuit board. The design has changed through the years, and some variations include other parts as well. But on a basic board, you're likely to find the following pieces:

A number of pins, which are used to connect with various components you might want to use with the Arduino. These pins come in two varieties: Digital pins, which can read and write a single state, on or off. Most Arduinos have 14 digital I/O pins. Analog pins, which can read a range of values, and are useful for more fine-grained control. Most Arduinos have six of these analog pins. These pins are arranged in a specific pattern, so that if you buy an add-on board designed to fit into them, typically called a "shield," it should fit into most Arduino-compatible devices easily.

A power connector, which provides power to both the device itself, and provides a low voltage which can power connected components like LEDs and various sensors, provided their power needs are reasonably low. The power connector can connect to either an AC adapter or a small battery.

A microcontroller, the primary chip, which allows you to program the Arduino in order for it to be able to execute commands and make decisions based on various input. The exact chip varies depending on what type of Arduino you buy, but they are generally Atmel controllers, usually an ATmega8, ATmega168, ATmega328, ATmega1280, or ATmega2560. The differences between these chips are subtle, but the biggest difference a beginner will notice is the different amounts of onboard memory.

A serial connector, which on most newer boards is implemented through a standard USB port. This connector allows you to communicate to the board from your computer, as well as load new programs onto the device. Often times Arduinos can also be powered through the USB port, removing the need for a separate power connection.

A variety of other small components, like an oscillator and/or a voltage regulator, which provide important capabilities to the board, although you typically don't interact with these directly; just know that they are their medical devices, remote controls, office machines, appliances, power tools, toys etc.

Writing codes for Arduino:

Arduino IDE:

To edit, compile and upload software projects (sketches) from your computer, you need at least the Arduino IDE. Two other pieces of software are also recommended, i.e. Processing for the connection between your Arduino projects and your computer and Fritzing to create drawings. This chapter describes the Arduino IDE, Processing and Fritzing are described in the following chapters. With this Arduino Integrated Development Environment, you can edit, compile and upload Arduino sketches to the Arduino boards. In this document I refer to version 1.6.5, available for Windows, OS X and Linux (and an older version for Raspberry Pi).

Step 1: Download and Install the IDE

You can download the IDE from the official Arduino website. Since the Arduino uses a USB to serial converter (which allow it to communicate with the host computer), the Arduino board is compatible with most computers that have a USB port. Of course, you will need the IDE first. Luckily, the Arduino designers have released multiple versions of the IDE for different operating systems, including Windows, Mac, and Linux. In this tutorial, we will use Window 10, so ensure that you download the correct version of the IDE if you do not have Windows 10.

.

Fig 4: Downloading Arduino IDE

Once downloaded, install the IDE and ensure that you enable most (if not all) of the options, including the drivers.

Step 2: Get the Arduino COM Port Number

Next, you'll need to connect the Arduino Uno board to the computer. This is done via a USB B connection. Thanks to the wonderful world of USB, we do not need to provide power to the Arduino, as the USB provides 5V up to 2A. When the Arduino is connected, the operating system should recognize the board as a generic COM port (for example, my

Arduino Uno uses a CH340G, which is an RS-232 serial to USB converter). Once it's recognized, we will need to find out what port number it has been assigned. The easiest way to do this is to type "device manager" into Windows Search and select Device Manager when it shows.

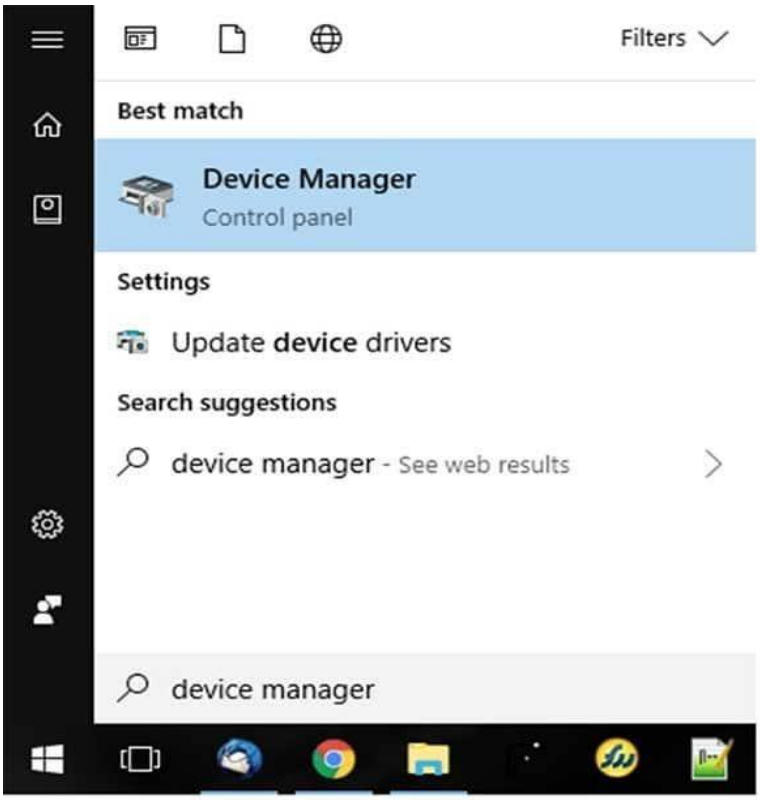


Fig 5: Selecting COM Port

In the Device Manager window, look for a device under "Ports (COM & LPT)", and chances are the Arduino will be the only device on the list. In my Device Manager, the Arduino shows up as COM7 (I know this because CH340 is in the device name).

Fig 6: COM Port

Be warned, the Arduino won't always be recognized automatically. If your Arduino is not recognized. then uninstall the driver, remove the Arduino, reinsert the Arduino, find the unrecognized device, right click "Update driver", and then click "Search automatically". This should fix 99 out of 100 problems. Windows can be a real pain sometimes with COM ports, as it can magically change their numbers between connections. In other words, one day, your Arduino may be on port 7 (as shown here), but then on other days, Windows may shift it to a different port number. As I understand it, this happens when you connect other COM ports to your system (which I do frequently).So, if you can't find your Arduino on the port that you usually use, just go to your Device Manager and check what port it's actually on and, if necessary, update your driver.

Step 3: Configure the IDE:

Now that we have determined the COM port that the Arduino is on, it's time to load the Arduino IDE and configure it to use the same device and port. Start by loading the IDE. Once it's loaded, navigate to Tools > Board > Arduino Uno. However, if you are using a different board (i.e., not the Arduino Uno), you must select the proper board!

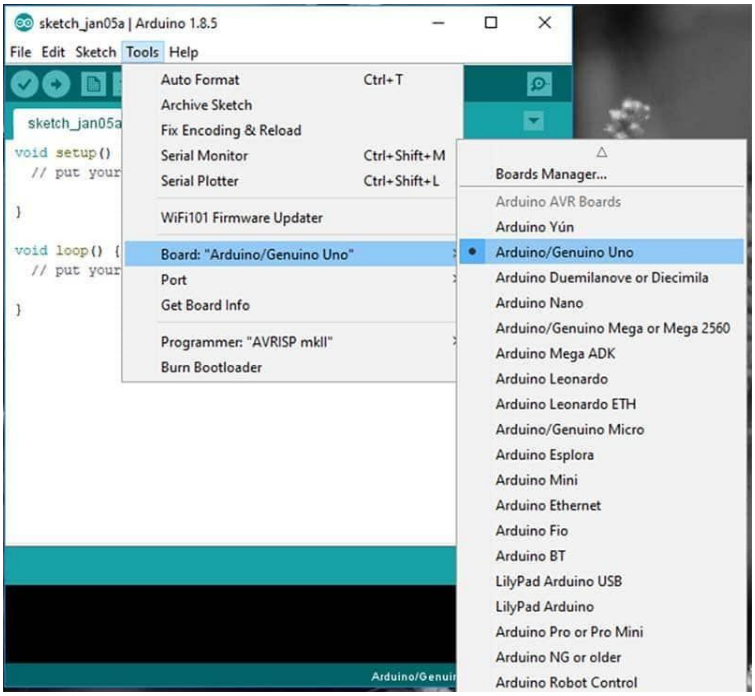


Fig 7: Selecting the Board

Next, you must tell the IDE which COM port the Arduino is on. To do this, navigate to Tools. Port > COM7. Obviously, if your Arduino is on a different port, select that port instead

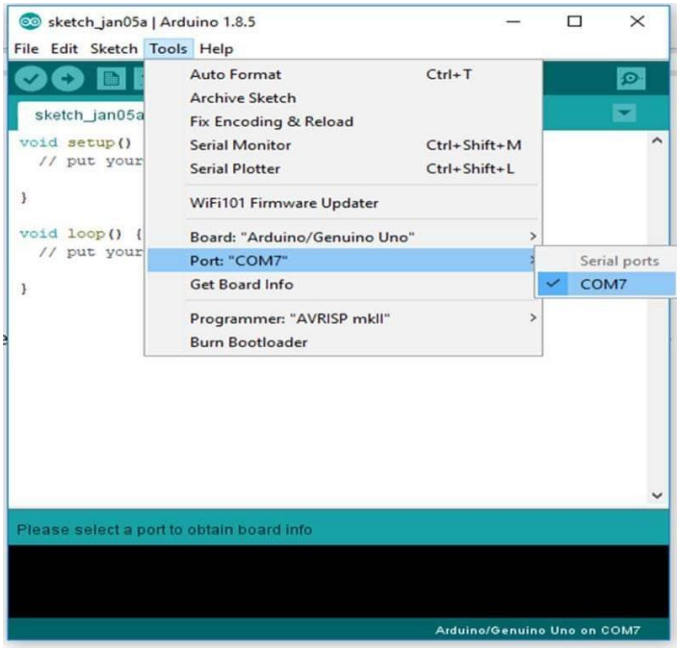


Fig 8: Select COM7

Step 4: Loading a Basic Example:

For the sake of simplicity, we will load an example project that the Arduino IDE comes with. This example will make the onboard LED blink for a second continuously. To load this example, click File > Examples > 01. Basics > Blink. instead.

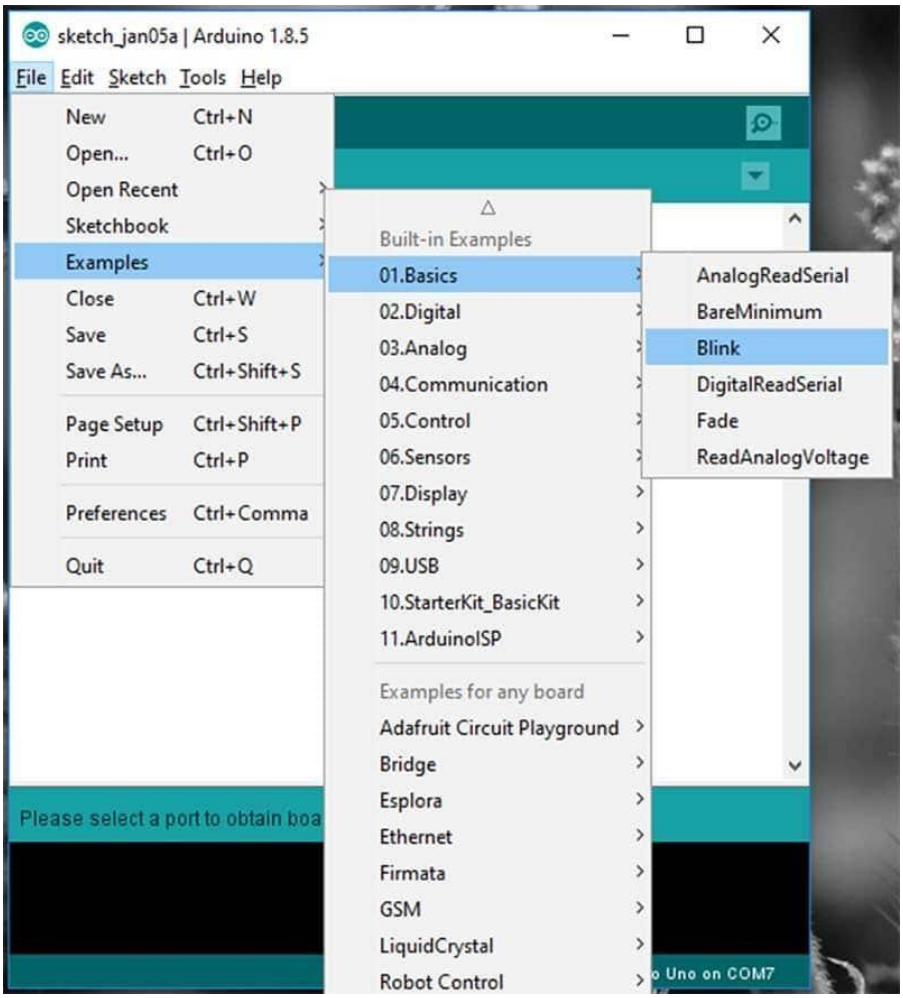


Fig 9: Example for IDE

With the example loaded, it's time to verify and upload the code. The verify stage checks the code for errors, then compiles the ready-for-uploading code to the Arduino. The upload stage actually takes the binary data, which was created from the code, and uploads it to the Arduino via the serial port.

To verify and compile the code, press the check mark button in the upper left window.

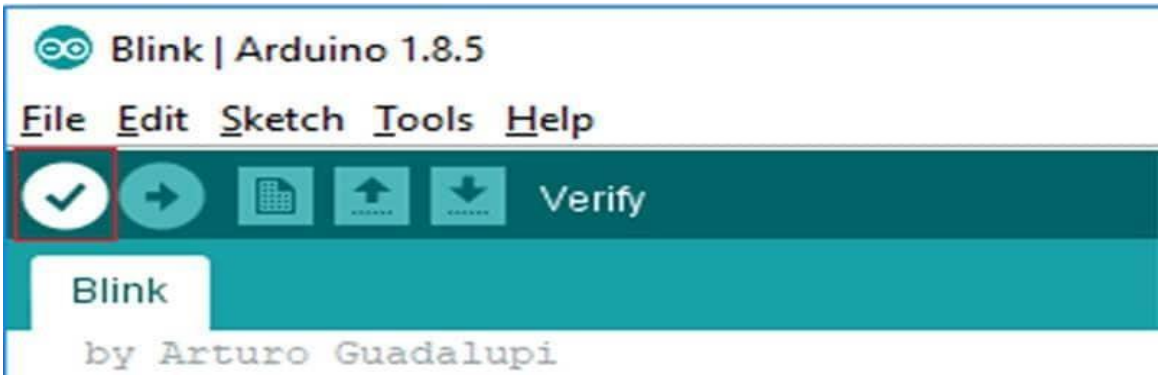


Fig 10: Compiling the code

If the compilation stage was successful, you should see the following message in the output window at the bottom of the IDE. You might also see a similar message—just it's one that does not have words like "ERROR" and "WARNING".

With the code compiled, you must now upload it the Arduino Uno. To do this, click the arrow next to the check mark.

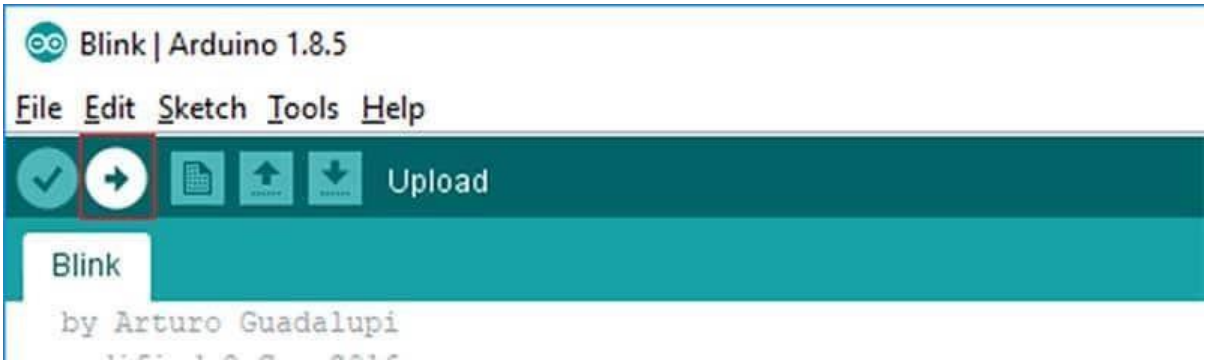


Fig 11: Execution of the code

2.3.2 DHT 11 SENSOR:

Humidity is the measure of water vapors present in the air. The level of humidity in air affects various physical, chemical and biological processes. In industrial applications, humidity can affect the business cost of the products, health and safety of the employees. So, in semiconductor industries and control system industries measurement of humidity is very important. Humidity measurement determines the amount of moisture present in the gas that can be a mixture of water vapors, nitrogen, argon or pure gas etc. Humidity sensors are of two types based on their measurement units. They are a relative humidity sensor and Absolute humidity sensor. DHT11 is a digital temperature and humidity sensor.

The DHT11 is a commonly used **Temperature and humidity sensor**. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.

The sensor can measure temperature from 0°C to 50°C and humidity from 20% to 90% with an accuracy of ±1°C and ±1%. So, if you are looking to measure in this range then this sensor might be the right choice for you. The DHT11 is a basic, low cost digital temperature and humidity sensor.

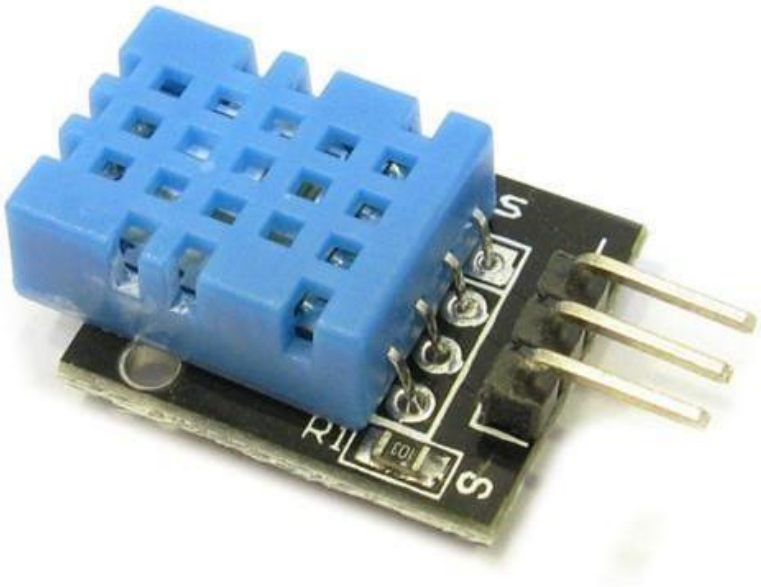


Fig. 11: DHT 11 Sensor

DHT11 is a single wire digital humidity and temperature sensor, which provides humidity and temperature values serially with one-wire protocol.DHT11 sensor provides relative humidity value in percentage (20 to 90% RH) and temperature values in degree Celsius (0 to 50 °C).DHT11 sensor uses resistive humidity measurement component, and NTC temperature measurement component. Microcontrollers.

DHT11 SPECIFICATIONS:

- 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%

Pin Configuration:

Pin Number	Pin Name	Description
1	Vcc	Connect to +5V or +3.3V supply voltage
2	Data	Digital output pin
3	Vcc	Not in use
4	Ground	Connected to the ground of the system

Communication with DHT 11:

DHT11 uses only one wire for communication. The voltage levels with certain time value defines the logic one or logic zero on this pin.

The communication process is divided in three steps, first is to send request to DHT11 sensor then sensor will send response pulse and then it starts sending data of total 40 bits to the microcontroller.

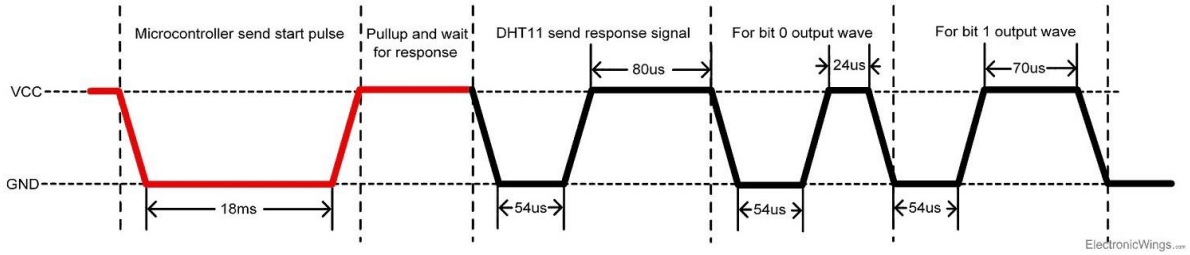


Fig 13: Communication Process

RESPONSE:

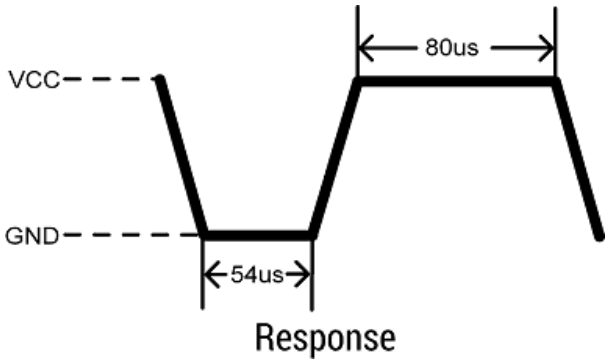


Fig 14: Response of DHT11

After getting start pulse from, DHT11 sensor sends the response pulse which indicates that DHT11 received start pulse. The response pulse is low for 54us and then goes high for 80us.

DATA:

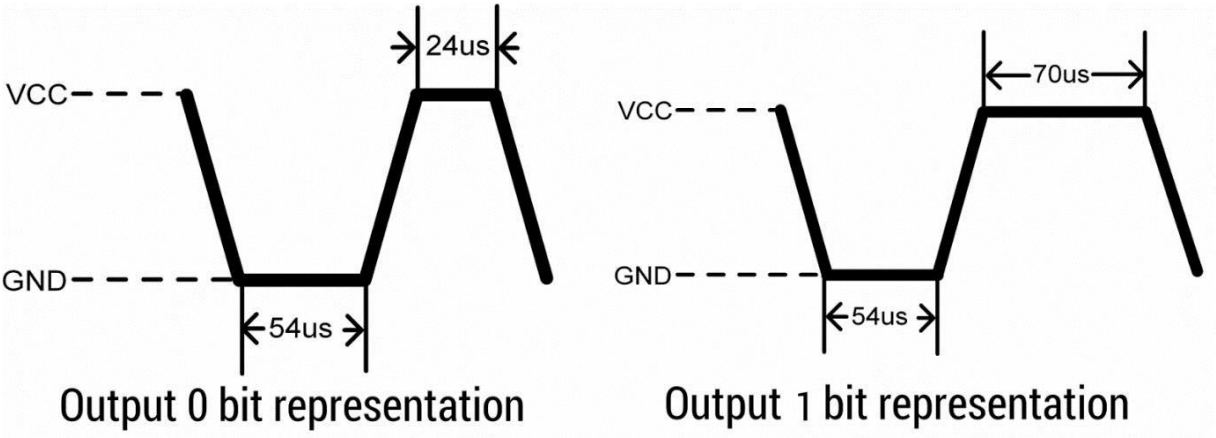


Fig 15: Data bit representation of DHT11

- After sending the response pulse, DHT11 sensor sends the data, which contains humidity and temperature value along with checksum.
- The data frame is of total 40 bits long, it contains 5 segments (byte) and each segment is 8-bit long.
- In these 5 segments, first two segments contain humidity value in decimal integer form. This value gives us Relative Percentage Humidity. 1st 8-bits are integer part and next 8 bits are fractional part.
- Next two segments contain temperature value in decimal integer form. This value gives us temperature in Celsius form.
- Last segment is the checksum which holds checksum of first four segments.
- Here checksum byte is direct addition of humidity and temperature value. And we can verify it, whether it is same as checksum value or not. If it is not equal, then there is some error in the received data.
- Once data received, DHT11 pin goes in low power consumption mode till next start pulse.

End of Frame:

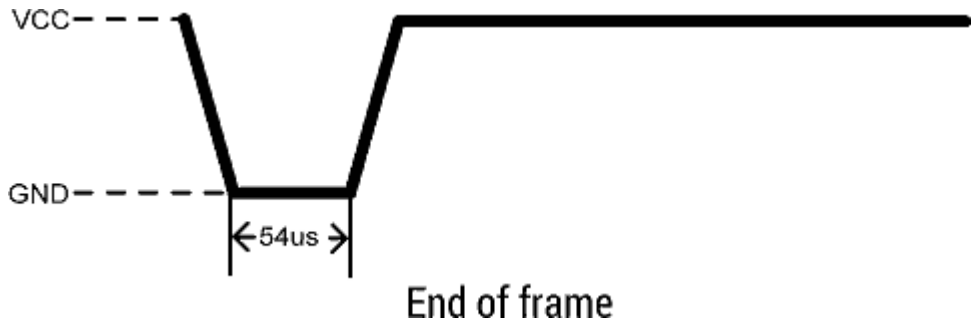


Fig 16: Ending bit of DHT11

After sending 40-bit data, DHT11 sensor sends 54us low level and then goes high. After this DHT11 goes in sleep mode.

Applications:

1. Measure temperature and humidity
2. Local Weather station
3. Automatic climate control
4. Environment monitoring

2.3.3 GSM MODULE:

GSM is a mobile communication modem; it is stands for global system for mobile communication (GSM). The idea of GSM was developed at Bell Laboratories in 1970. It is widely used mobile communication system in the world. GSM is an open and digital cellular technology used for transmitting mobile voice and data services operates at the 850MHz, 900MHz, 1800MHz and 1900MHz frequency bands.

.

Fig 17: GSM Module

GPRS:

GPRS or General Packet Radio Service is an extension of the GSM Network. GPRS is an integrated part of the GSM Network which provides an efficient way to transfer data with the same resources as GSM Network.

Originally, the data services (like internet, multimedia messaging etc.) in the GSM Network used a circuit – switched connection. In this type, the access time for the network are long and the charges for the data were based on the connection time. Also, this type of connection is not suitable for transmitting bursts of data. With the integration of GPRS, a packet – switching based data service, in to the GSM Network, the scene of data services has changes. In GPRS based packet – switching networks, the user device doesn't hold the

resources for a continuous time but efficiently uses a common pool. The access time in GPRS is very small and the main advantage is that it allows bursts of data to be transmitted. Also, the charges for data are based on the usage and not on the connection time.

GSM Architecture:

A GSM network consists of the following components:

1. A Mobile Station:

It is the mobile phone which consists of the transceiver, the display and the processor and is controlled by a SIM card operating over the network.

2. Base Station Subsystem:

It acts as an interface between the mobile station and the network subsystem. It consists of the Base Transceiver Station which contains the radio transceivers and handles the protocols for communication with mobiles. It also consists of the Base Station Controller which controls the Base Transceiver station and acts as an interface between the mobile station and mobile switching center.

3. Network Subsystem:

It provides the basic network connection to the mobile stations. The basic part of the Network Subsystem is the Mobile Service Switching Centre which provides access to different networks like ISDN, PSTN etc. It also consists of the Home Location Register and the Visitor Location Register which provides the call routing and roaming capabilities of GSM. It also contains the Equipment Identity Register which maintains an account of all the mobile equipment's wherein each mobile is identified by its own IMEI number. IMEI stands for International Mobile Equipment Identity.

Features of GSM Module:

1. Improved spectrum efficiency
2. International roaming
3. Compatibility with integrated services digital network (ISDN)
4. Support for new services.
5. SIM phonebook management
6. Fixed dialing number (FDN)
7. High-quality speech
8. Uses encryption to make phone calls more secure

9. Short message service (SMS)

AT COMMANDS OF GSM MODULE:

Based on the operation performed by the AT Commands, they are again divided in to four types: Test Commands, Read Commands, Write Commands and Execution Commands. We will now see the definition and syntax of all these command types.

Test Commands:

These commands are used to test whether the command exists (supported by the GSM GPRS Module) or not and also checks for the range of a command's sub parameters. When a Test Command is given to the GSM GPRS Module, it returns the list of all the parameters and also the range set of the parameters.

The syntax of a Test Command is ATCMD1=?<CR>

Example for Test Command: AT+CGMI=? (Request manufacturer identification)

Read Commands:

Read Commands will return the current value of the parameter. Using these commands, we can read the current settings of the GSM GPRS Module.

The syntax for Read Commands is: ATCMD1?<CR>

Example for Read Command: AT+CSCA? (Query for Service Centre)

Write or Set Commands:

Set Commands will attempt to change or modify the settings of the GSM GPRS Module by setting a new parameter for the particular command.

The syntax for Set Commands is: ATCMD1=value1, value2, value3…valuen<CR>

An example for Set Command: AT+CMGF=1 (Set Message format to TEXT Mode).

Execution Command:

These commands perform an operation like send an SMS, retrieving information about battery charging status etc. They read the non – variable sub parameters that are affected by the GSM Module.

Syntax of Execution Commands: ATCMD1<CR>

Example for Execution Commands: AT+CMGS=<number><CR> <text message> <CTRL-

Z> (Sends text message to the number).

Information Responses and Final Codes

After sending the AT Commands to the GSM GPRS Module, we have look for the response. For example, if we send the command as AT+CGMI<CR> to the GSM Module, then the response would be as follows.

<CR><LF>Apple<CR><LF>

<CR><LF>OK<CR><LF>

Here, <CR> is Carriage Return and <LF>is Line Feed.

In a HyperTerminal, if you entered AT+CGMI<CR>, the response will look something like this.

AT+CGMI <– Command entered

Apple <– Information Response

OK <– Final Code

The syntax of the information response and final command is as follows:

<Carriage Return><Line Feed> <Information Response / Final Result Code> <Carriage Return><Line Feed>

<CR><LF><Response><CR><LF>

NOTE: The sequence of execution of commands will be first commands, then second command, followed by the rest i.e. a sequential execution of commands.

If there is an error anywhere in the execution, an error code is returned by the GSM Module and the execution of further commands is terminated.

Frequently used AT Commands:

In this list, you can find out some of the most commonly used AT Commands. For a complete list of AT Commands and their definitions, it is advised to refer the manufacturer data. The <Carriage Return> or <CR> is denoted by this symbol ↲.

To check the communication between the GSM Module and the host (Computer)

AT ↲

OK

To make a voice call

ATD9848032919; ↲

To answer or receive an incoming call

ATA ↲

To redial the last number

ATDL ↲

To disconnect a call

ATH ↲

To set the message mode to text mode

AT+CMGF=1 ↲

OK

To send a text message

AT+CMGS=”9848032919” ↲

Message

CTRL+Z

2.3.4 GPS Module:

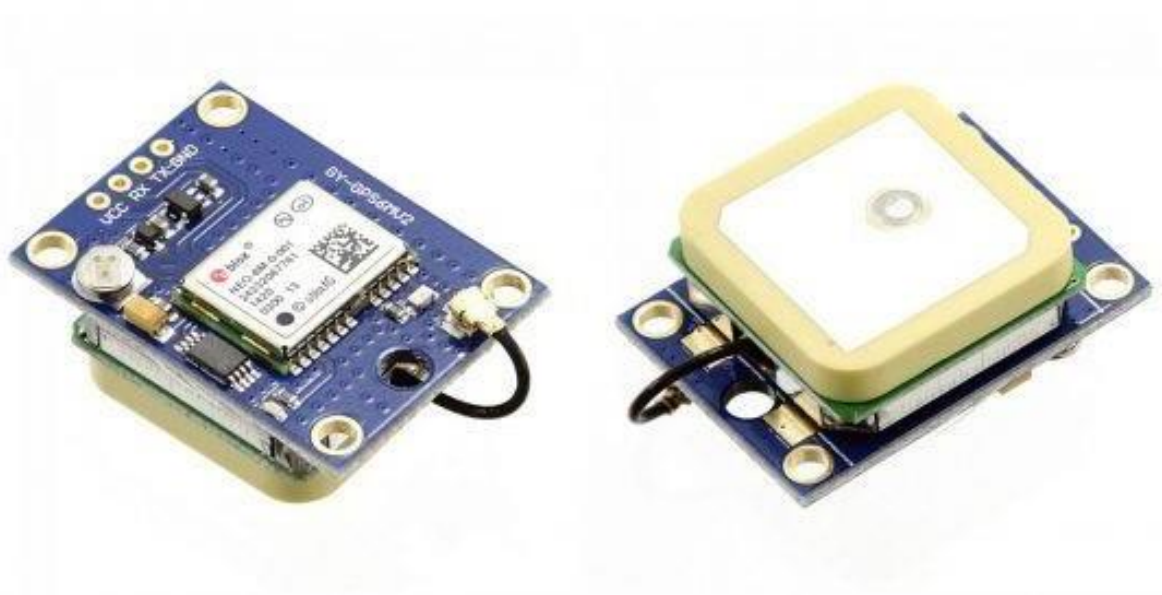


Fig 18: GPS Module

The u-blox NEO-6M GPS engine on these modules is quite a good one, and it also has high sensitivity for indoor applications. Furthermore, there's one MS621FE-compatible rechargeable battery for backup and EEPROM for storing configuration settings. The module works well with a DC input in the 3.3- to 5-V range (thanks to its built-in voltage regulator). The original circuit diagram of the module, borrowed from the web, is shown below

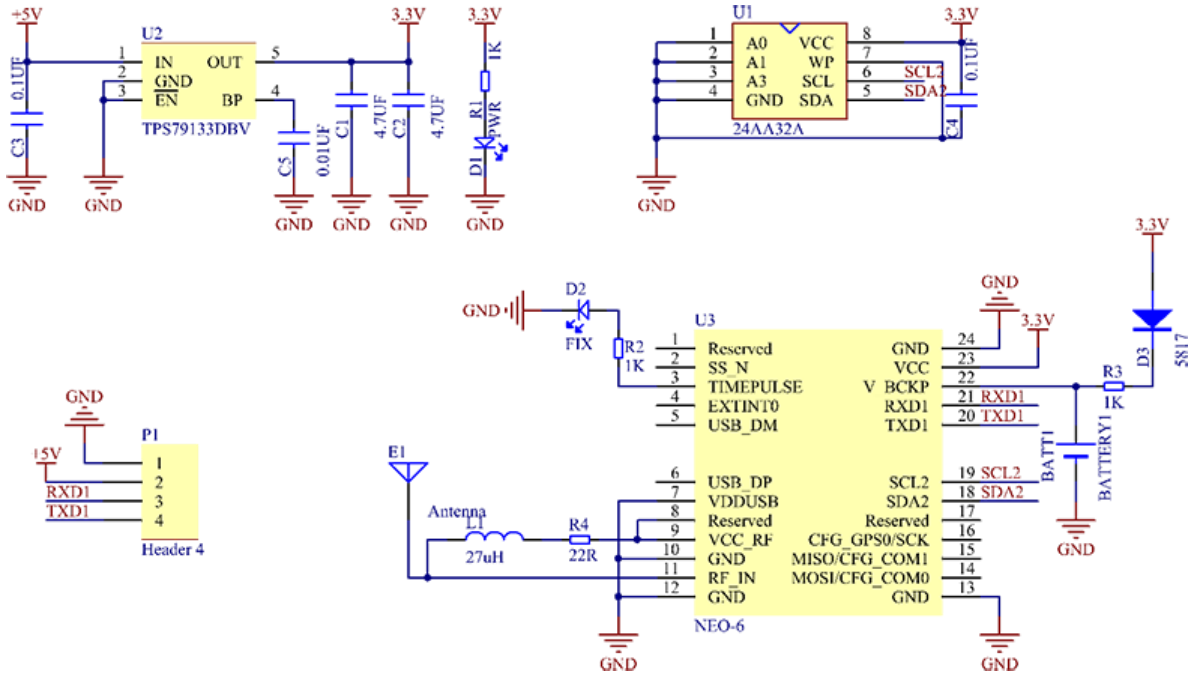


Fig 19: Pin diagram of GPS Module

As indicated, the GPS modules are based on the u-blox NEO-6M GPS engine. The type number of the NEO-6M is NEO-6M-0-001, and its ROM/FLASH version is ROM 7.0.3 (PCN reference UBX-TN-11047-1). The NEO-6M module includes one configurable UART interface for serial communication, but the default UART (TTL) baud rate here is 9,600. Because the GPS signal is right-hand circular-polarized (RHCP), the style of the GPS antenna will be different from the common whip antennas used for linear polarized signals. The most popular antenna type is the patch antenna. Patch antennas are flat, generally have a ceramic and metal body, and are mounted on a metal base plate. They are often cast in a housing. For more information about u-blox reference designs, see their website. Remember, the position of the antenna mounting is very crucial for optimal performance of the GPS receiver. When using the patch antenna, it should be oriented parallel to the geographic horizon. The antenna must have full view of the sky, ensuring a direct line of sight with as many visible satellites.

FEATURES AND ELECTRICAL CHARACTERISTICS:

- Standalone GPS receiver
- Anti-jamming technology
- UART Interface at the output pins (Can use SPI ,I2C and USB by soldering pins to the chip core)
- Under 1 second time-to-first-fix for hot and aided starts
- Receiver type: 50 Channels - GPS L1 frequency - SBAS (WAAS, EGNOS, MSAS, GAGAN)
- Time-To-First-fix: For Cold Start 32s, For Warm Start 23s, For Hot Start <1s
- Maximum navigation update rate: 5Hz
- Default baud rate: 9600bps
- EEPROM with battery backup
- Sensitivity: -160dBm
- 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

APPLICATIONS:

1. GPS application
2. Smart phone and tablets
3. Navigation systems
4. Drones
5. Hobby projects

2.3.5 RF Module:

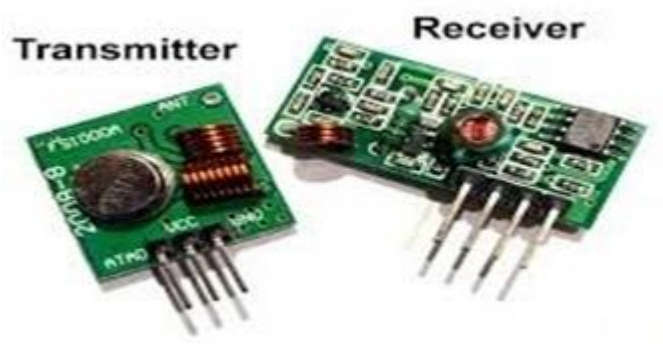


Fig 20: RF Module

An RF module is a small electronic device used to transmit and/or receive radio signals between two devices. In an embedded system it is often desirable to communicate with another device wirelessly. This wireless communication may be accomplished through optical communication or through radio-frequency (RF) communication. For many applications, the medium of choice is RF since it does not require line of sight. RF communications incorporate a transmitter and a receiver. They are of various types and ranges. Some can transmit up to 500 feet. RF modules are typically fabricated using RF CMOS technology.

RF modules are widely used in electronic design owing to the difficulty of designing radio circuitry. Good electronic radio design is notoriously complex because of the sensitivity of radio circuits and the accuracy of components and layouts required to achieve operation on a specific frequency. In addition, reliable RF communication circuit requires careful monitoring of the manufacturing process to ensure that the RF performance is not adversely affected. Finally, radio circuits are usually subject to limits on radiated emissions, and require Conformance testing and certification by a standardization organization such as ETSI or the U.S. Federal Communications Commission (FCC). For these reasons, design engineers will often design a circuit for an application which requires radio communication and then "drop in" a pre-made radio module rather than attempt a discrete design, saving time and money on development.

RF modules are most often used in medium and low volume products for consumer applications such as garage door openers, wireless alarm or monitoring systems, industrial

remote controls, smart sensor applications, and wireless home automation systems. They are sometimes used to replace older infrared communication designs as they have the advantage of not requiring line-of-sight operation.

Several carrier frequencies are commonly used in commercially available RF modules, including those in the industrial, scientific and medical (ISM) radio bands such as 433.92 MHz, 915 MHz, and 2400 MHz's These frequencies are used because of national and international regulations governing the use of radio for communication. Short Range Devices may also use frequencies available for unlicensed such as 315 MHz and 868 MHz's

TYPES OF RF MODULE:

The term RF module can be applied to many different types, shapes and sizes of small electronic sub assembly circuit board. It can also be applied to modules across a huge variation of functionality and capability. RF modules typically incorporate printed circuit board, transmit or receive circuit, Antenna and serial interface for communication to the host processor.

Most standard, well known types are covered here:

1. Transmitter module
2. Receiver module
3. Transceiver module
4. System on chip module.
5. Transmitter modules

An RF transmitter module is a small PCB and sub-assembly capable of transmitting a radio wave and modulating that wave to carry data. Transmitter modules are usually implemented alongside a microcontroller which will provide data to the module which can be transmitted. RF transmitters are usually subject to regulatory requirements which dictate the maximum allowable transmitted power, harmonics, and band edge requirements.

Receiver modules:

An RF receiver module receives the modulated RF signal, and demodulates it. There are two types of RF receiver modules: super heterodyne receivers and super regenerative module. Super regenerative modules are usually low cost and low power designs using a series of amplifiers to extract modulated data from a carrier wave. Super regenerative

modules are generally imprecise as their frequency of operation varies considerably with temperature and power supply voltage. Super heterodyne receivers have a performance advantage over super regenerative; they offer increased accuracy and stability over a large voltage and temperature range. This stability comes from a fixed crystal design which in the past tended to mean a comparatively more expensive product. However, advances in receiver chip design now mean that currently there is little price difference between super heterodyne and super regenerative receiver modules.

TRANSCEIVER MODULE:

An RF transceiver module incorporates both a transmitter and receiver. The circuit is typically designed for half duplex operation, although full duplex modules are available, typically at a higher cost due to the added complexity.

SYSTEM ON A CHIP (SOC) MODULE:

An SoC module is the same as a transceiver module, but it is often made with an onboard microcontroller. The microcontroller is typically used to handle radio data packetization or managing a protocol such as an IEEE 802.15.4 compliant module. This type of module is typically used for designs that require additional processing for compliance with a protocol when the designer does not wish to incorporate this processing into the host microcontroller.

MODULE PHYSICAL CONNECTION:

A variety of methods are used to attach an RF module to a printed circuit board, either with through-hole technology or surface-mount technology. Through-hole technology allows the module to be inserted or removed without soldering. Surface-mount technology allows the module to be attached to the PCB without an additional assembly step. Surface-mount connections used in RF modules include land grid array (LGA) and castellated pads. The LGA package allows for small module sizes as the pads are all beneath the module but connections must be X-rayed to verify connectivity. Castellated Holes enable optical inspection of the connection but will make the module footprint physically larger to accommodate the pads.

WIRELESS PROTOCOL USED IN RF MODULE:

RF modules, especially SoC modules, are frequently used to communicate according to a pre-defined wireless standard, including:

1. Zigbee
2. Bluetooth Low Energy
3. Wi-Fi
4. IEEE 802.15.4
5. Z-Wave
6. Wirepas

However, RF modules also frequently communicate using proprietary protocols, such as those used in garage door openers

TYPICAL APPLICATIONS:

1. Vehicle monitoring
2. Remote control
3. Telemetry
4. Small-range wireless network
5. Wireless meter reading
6. Access control systems
7. Wireless home security systems
8. Area paging
9. Industrial data acquisition system
10. Radio tags reading
11. RF contactless smart cards
12. Wireless data terminals
13. Wireless fire protection systems
14. Biological signal acquisition
15. Hydrological and meteorological monitoring
16. Robot remote control
17. Wireless data transmissions
18. Digital video/audio transmission

19. Digital home automation, such as remote light/switch
20. Industrial remote control, telemetry and remote sensing
21. Alarm systems and wireless transmission for various types of low-rate digital signal
22. Remote control for various types of household appliances and electronics projects
23. Many other applications fields related to RF wireless controlling
24. Mobile web server for elderly people monitoring

2.4 SOFTWARE REQUIREMENT:

The software's which we are using in this project are Arduino IDE.

2.4.1 ARDUINO IDE SOFTWARE:

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

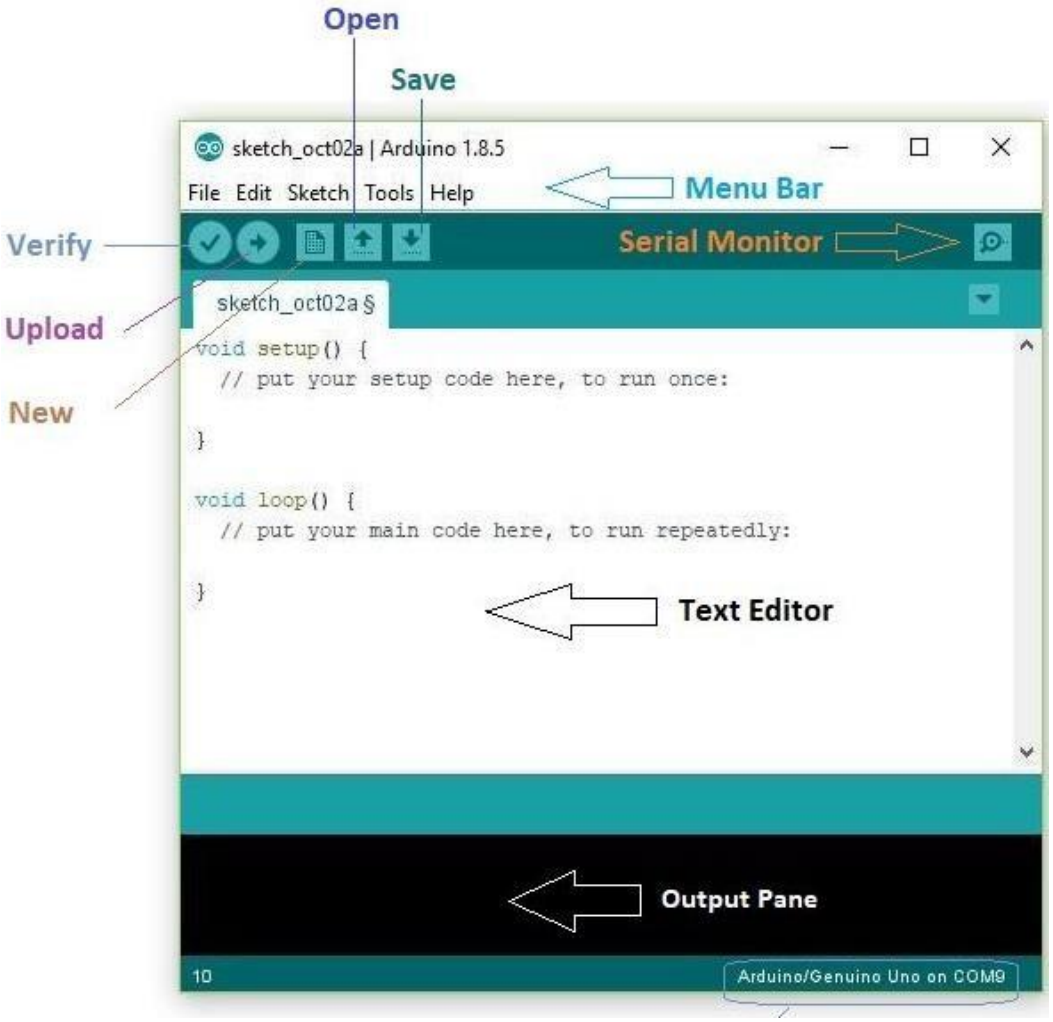


Fig 21: Menu Bar of Arduino IDE

The bar appearing on the top is called **Menu Bar** that comes with five different options as follow

- **File** – You can open a new window for writing the code or open an existing one.
- **Edit** – Used for copying and pasting the code with further modification for font
- **Sketch** – For compiling and programming.
- **Tools** – Mainly used for testing projects. The Programmer section in this panel is used for burning a bootloader to the new microcontroller.
- **Help** – In case you are feeling sceptical about software, complete help is available from getting started to troubleshooting.
- The button appearing on the top right corner is a **Serial Monitor** – A separate pop-up window that acts as an independent terminal and plays a vital role for sending and

receiving the Serial Data. You can also go to the Tools panel and select Serial Monitor, or pressing Ctrl+Shift+M all at once will open it instantly. The Serial Monitor will actually help to debug the written Sketches where you can get a hold of how your program is operating. Your Arduino Module should be connected to your computer by USB cable in order to activate the Serial Monitor.

- You need to select the baud rate of the Arduino Board you are using right now. For my Arduino Uno Baud Rate is 9600, as you write the following code and click the Serial Monitor, the output will show as the image below.

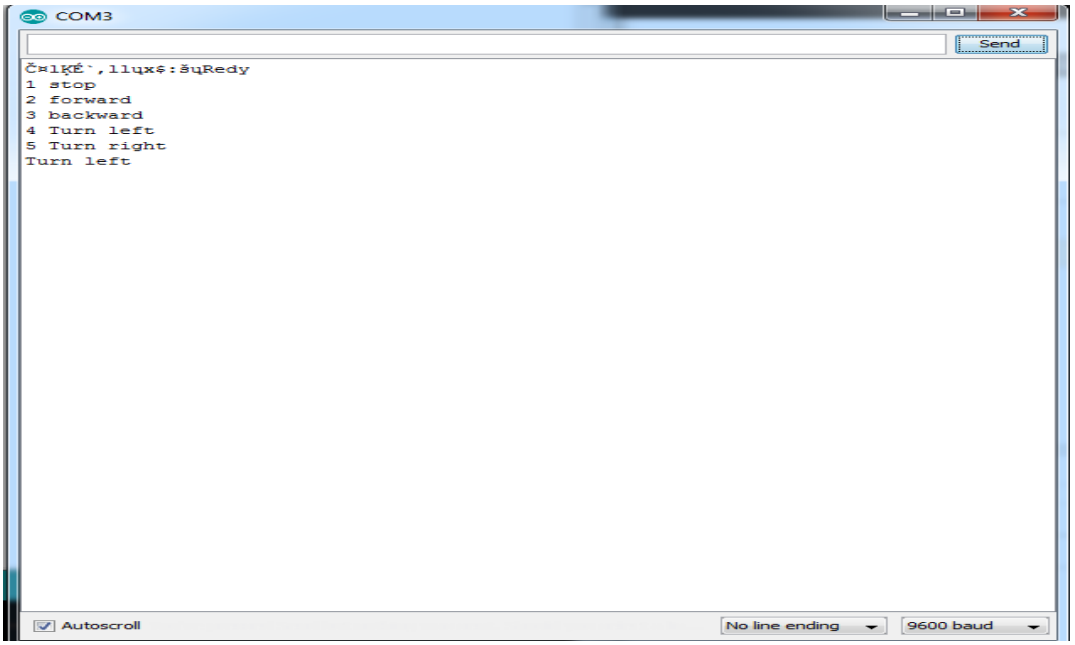


Fig 22: Serial Monitor of IDE

CHAPTER – 3

PROCEDURE

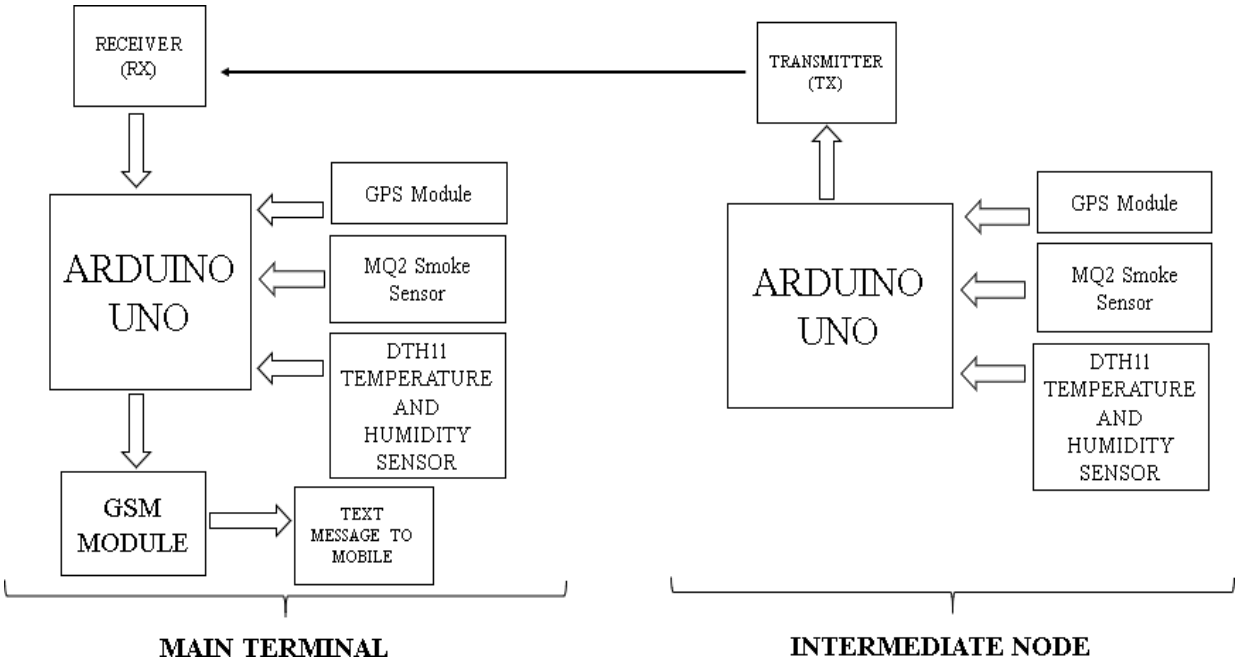
3.1 Steps Involved:

- Firstly, we have written a code interfacing Arduino and GSM module using Arduino IDE
- Then we have modified the code so that we can also interface GPS module and RF module to Arduino
- Later we compiled it and checked the output on serial monitor
- Connect the DHT11 Temperature and Humidity sensor to Arduino
- Connecting power supplies to all the Arduinos
- Observe the data recorded by DHT11 which includes Temperature and Air Quality
- If there's a fire near any of the node then the GSM Module will send an SMS containing the condition of fire and also a link to the geographical location of fire
- Verifying the data and coming to a conclusion

3.2 HARDWARE SETUP:

The hardware setup involves interfacing of 3 major components to Arduino. Firstly, the GPS Module should be connected to an Arduino (Main Terminal). GSM module is used only in the main terminal so that SMS alerts can be received by us. Secondly, GPS Module should be interfaced to both the Arduinos so that in case of detection of fire, a link containing the geographical location can be obtained so as to know where the fire is in the forest. A DHT11 temperature and humidity sensor is connected to both the Arduinos. The third major component is RF Module which is used to communicate between the tree nodes so as to find if the fire is detected on other parts of fire as well. This is done by connecting the transmitter to tree nodes and the receiver to the main node. The block diagram is shown below.

BLOCK DIAGRAM OF THE PROJECT:



The forest fire detection module works in three different stages. The first stage consists of reading some external environmental parameters like temperature and smoke. The first stage is done with the help of some sensors which are used to sense and convert analog data to digital data. The sensors read parameters like temperature, humidity and air quality then send this information to the next nearest node. This process goes on until the information reaches to the final node or the main terminal which is the second stage of the overall process. The third stage consists of transmission of the information to the forest fire monitoring unit. Each node has a temperature and humidity sensor, a smoke sensor and microcontroller unit. Arduino has been used as the microcontroller device. The sensors interact with the Arduino and store the information for comparison process. There is a predefined threshold value to each of these parameters. The microprocessor compares the sensor values at regular intervals of times with the threshold values. Based on the comparison if the input values of sensors exceed the threshold the node transmits the information to the next nearby node which again in turn transmits the information to the other nearby node. In this way the message flow is regulated in this model.

CIRCUIT DIAGRAM OF THE PROJECT:

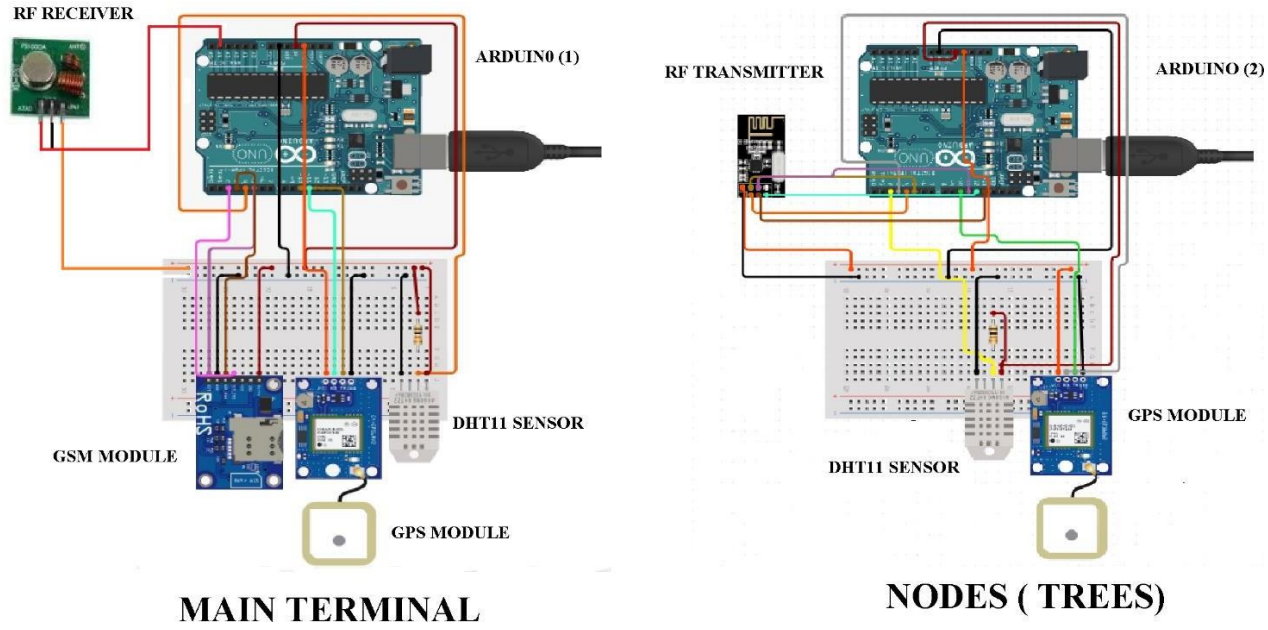


Fig 23: Circuit Diagram

3.3 SOFTWARE SETUP:

- Open Arduino IDE
- Write down the code given below
- Compile the code and check for errors, if any then rectify
- Run the code and Arduino

3.3.1 ARDUINO CODE FOR FOREST FIRE DETECTION SYSTEM WITH SMS ALERTS:

```
// (PART-1) Receiver Side Arduino CODE
// Include RadioHead Amplitude Shift Keying Library
#include <SoftwareSerial.h>
```

```
#include <RH_ASK.h>
// Include dependant SPI Library
#include <SPI.h>
#include <DHT.h>

SoftwareSerial mySerial(4, 3);

//Constants
#define DHTPIN 2     // what pin we're connected to
#define DHTTYPE DHT11  // DHT11

int smokeA0 = A5;
// Your threshold value
int sensorThres = 400;
// Define Variables

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino

//Variables
float hum; //Stores humidity value
float temp; //Stores temperature value

// Define output strings

String str_humid;
String str_temp;
String str_smk;
String str_out;

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;
```

```
void setup()
{
   pinMode(smokeA0, INPUT);
   dht.begin();
   // Initialize ASK Object
   rf_driver.init();
   // Setup Serial Monitor
   Serial.begin(9600);
   mySerial.begin(115200);
}

void loop()
{
   delay(2000);
   hum = dht.readHumidity();
   temp= dht.readTemperature();
   Serial.print("Reciever Humidity = ");
   Serial.print(hum);
   Serial.print('\n');
   Serial.print("Reciever Temperature = ");
   Serial.println(temp);

   int analogSensor = analogRead(smokeA0);
   String smk;
   // Checks if it has reached the threshold value
   if (analogSensor > sensorThres)
   {
     Serial.print("Smoke at Reciever");
     Serial.print('\n');
     smk = "Smoke";
   }
   else
   {
```

```
  Serial.print("Clean at Reciever");
  Serial.print('\n');
  smk = "Clean";
 }

 // Set buffer to size of expected message
 uint8_t buf[20];
 uint8_t buflen = sizeof(buf);
 // Check if received packet is correct size
if (rf_driver.recv(buf, &buflen))
{
 str_out = String((char*)buf);
 for (int i = 0; i < str_out.length(); i++)
 {
  if (str_out.substring(i, i+1) == ",")
    {
    str_humid = str_out.substring(0, i);
    str_temp = str_out.substring(i+1, i+6);
    str_smk = str_out.substring(i+7,i+12 );
    break;
    }
  }
 }

  // Print values to Serial Monitor
  Serial.print("Humidity: ");
  Serial.print(str_humid);
  Serial.print(" - Temperature: ");
  Serial.println(str_temp);
  Serial.print(" - Air Quality: ");
  Serial.println(str_smk);
```

```
if( hum >= 60 && temp >= 25 )
{
Serial.print("Fire Detected at Reciever ");
mySerial.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
delay(1000);  // Delay of 1000 milli seconds or 1 second
mySerial.println("AT+CMGS=\"+918744984131\"\r"); // Replace x with mobile number
delay(1000);
mySerial.println("    FIRE ALERT !! ");
Serial.println('\n');
mySerial.println("Fire at Reciever Node ");
Serial.println('\n');
mySerial.println("Temperature : " + String(temp));
Serial.print('\n');
mySerial.println("Humidity : " + String(hum));
Serial.print('\n');
mySerial.println("Air Quality : " + smk);
delay(100);
mySerial.println((char)26);// ASCII code of CTRL+Z
delay(1000);
}
if( str_humid.toInt() >= 60 && str_temp.toInt() >= 25 )
{
Serial.print("Fire Detected at Transmitter ");
mySerial.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
delay(1000);
mySerial.println("AT+CMGS=\"+918744984131\"\r"); // Replace x with mobile number
delay(1000);
mySerial.println("    FIRE ALERT !! ");
Serial.println('\n');
mySerial.println("Fire at Transmitter Node ");
Serial.println('\n');
mySerial.println("Temperature : " + str_temp);
Serial.print('\n');
```

```
      mySerial.println("Humidity : " + str_humid);
      Serial.print('\n');
      mySerial.println("Air Quality : " + String(str_smk));
      delay(100);
      mySerial.println((char)26);// ASCII code of CTRL+Z
      delay(1000);
      }

}

//(PART-2) Transmitter Side Arduino Code

#include <RH_ASK.h>
#include <SPI.h>
#include <DHT.h>

//Constants
#define DHTPIN 2     // what pin we're connected to
#define DHTTYPE DHT11  // DHT11

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino

int smokeA0 = A5;
// Your threshold value
int sensorThres = 400;
// Define Variables

float hum;    // Stores humidity value in percent
float temp; // Stores temperature value in Celcius
float smk;

// Define output strings
```

```
String str_humid;
String str_temp;
String str_smk;
String str_out;



// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

// Initialize DHT sensor for normal 16mhz Arduino

void setup() {
  dht.begin();
  pinMode(smokeA0, INPUT);
  // Initialize ASK Object
  rf_driver.init();

}

void loop()
{

    delay(2000);  // Delay so DHT-22 sensor can stabalize

    hum = dht.readHumidity();  // Get Humidity value
    temp= dht.readTemperature();  // Get Temperature value

    // Convert Humidity to string
    str_humid = String(hum);
    Serial.print(hum);
    // Convert Temperature to string
    str_temp = String(temp);
```

```
  Serial.print(temp);
  int analogSensor = analogRead(smokeA0);
  // Checks if it has reached the threshold value
  if (analogSensor > sensorThres)
  {
   str_smk = "1";
  }
  else
  {
   str_smk = "0";
  }

  // Combine Humidity and Temperature
  if(str_smk == "1")
  {
  str_out = str_humid + "," + str_temp + "," + "Smoke";
  }
  if(str_smk == "0")
  {
  str_out = str_humid + "," + str_temp+ "," + "Clean";
  }
  // Compose output character
  const char *msg = str_out.c_str();

  rf_driver.send((uint8_t *)msg, strlen(msg));
  rf_driver.waitPacketSent();

}
```

CHAPTER-4

RESULT AND ANALYSIS

4.1 RESULTS:

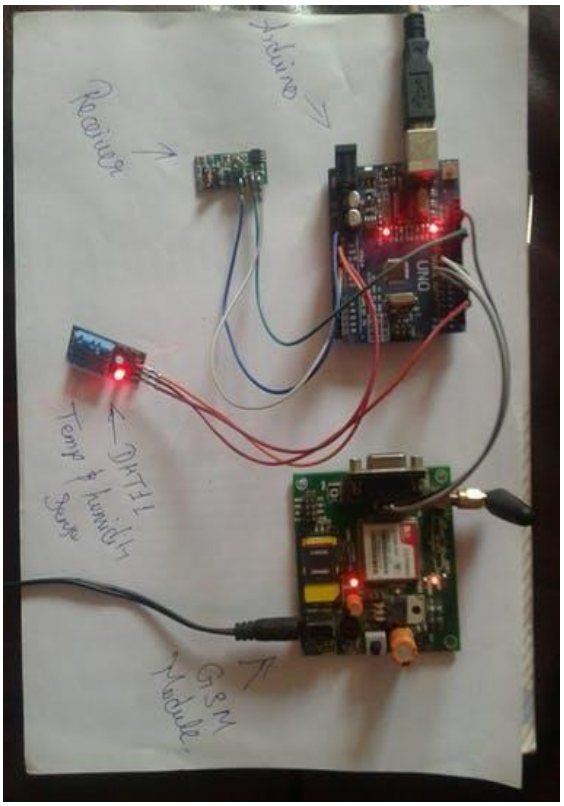


Fig 24: Main node setup

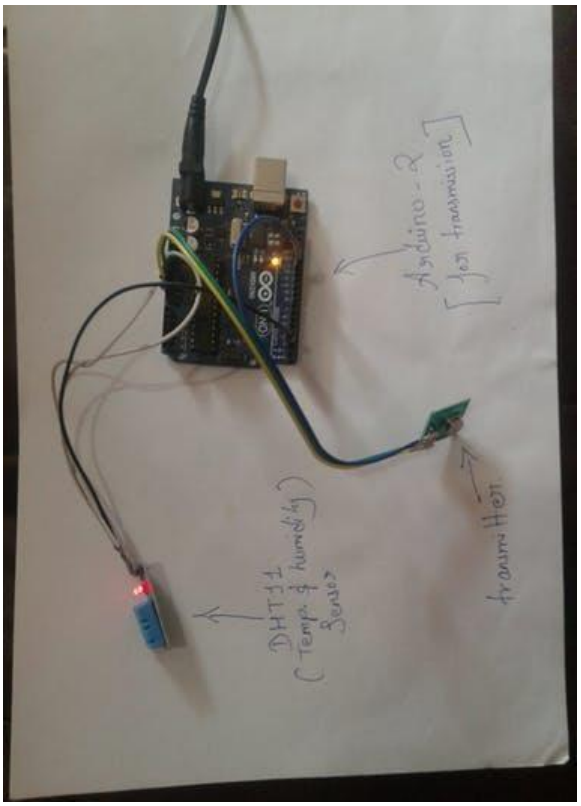


Fig 25: Tree node setup

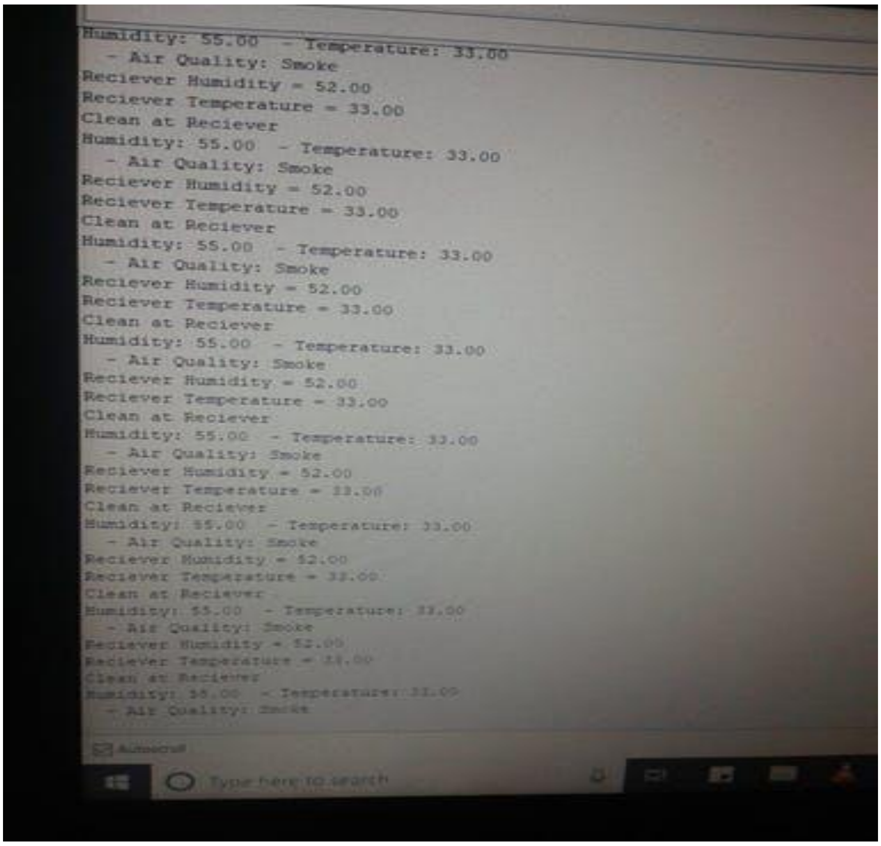


Fig 26: Result in Serial Monitor

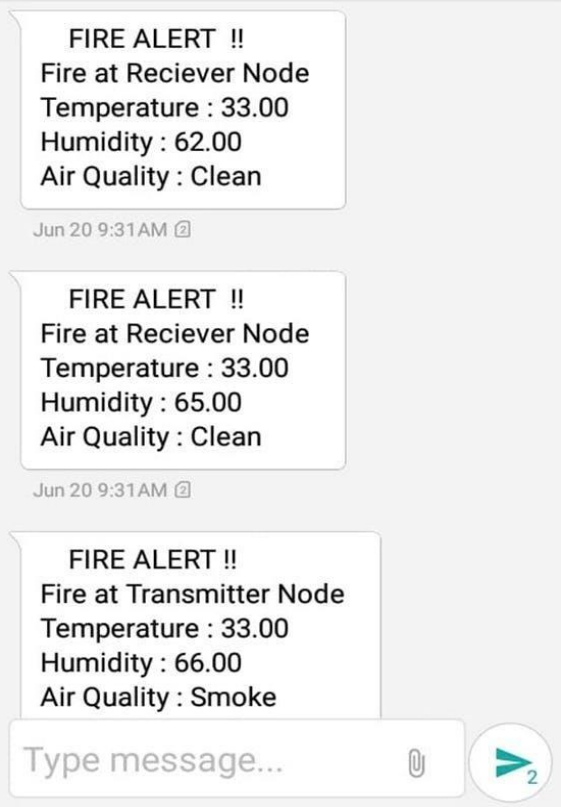


Fig 27: Result in mobile (SMS)

4.2 APPLICATIONS:

- The main purpose of this project was to apply this in the amazon bush fire incident.
- Can be used in Forests to create a large network of fire detecting system.
- Can also be used in buildings where each room has its own node.

4.3 CONCLUSION:

- It successfully detects fire and informs us using SMS.
- Geographical location of the source of fire is found out as each node has its own GPS.

- Communication is easier through RF communication as there is no assurance of internet availability in the depth of forest

- A large network of detection system can be created as the range of single RF module is around 800 meters.

4.4 FUTURE SCOPE:

The Project can be extended for addition of

1.A module by which water can be supplied so as to control the fire till the emergency team arrives.

2. Development of technology in order to reduce the number of microcontrollers used.

3. Developing a faster means of communication in order to cover the large area of detection system in forest

REFERENCNCES:

1. IoT Enabled Forest Fire Detection and Early Warning System by A. Divya ; T. Kavithanjali ; P. Dharshini (2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN))

2. Forest Fire Detection and Altering the Authorities by T. Saikumar, P. Sriramy (International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7, Issue-6S4, April 2019)

3. Research on Forest Fire Detection Based on Wireless Sensor Network Li Guang-Hui ; Zhao Jun ; Wang Zhi 2006 6th World Congress on Intelligent Control and Automation

4. J. Lloret, M. Garcia, D. Bri, and S. Sendra, “A wireless sensor network deployment for

rural and forest fire detection and verification," Sensors, vol. 9, no. 11, pp. 8722–8747, 2009.

5. P. Bahl and V. N. Padmanabhan, RADAR: An In-Building RF-Based User Location and Tracking System, In Proceedings of the IEEE INFOCOM '00, March 2000.

6. D. M. Doolin, S. D. Glaser and N. Sitar. Software Architecture for GPS-enabled Wild fire Sensorboard.TinyOS Technology Exchange, February 26, 2004, University of California, Berkeley CA. Google Scholar.

7. Method for fire and smoke detection in monitored forest areas David Asatryan ; Samvel Hovsepyan 2015 Computer Science and Information Technologies (CSIT)

8. R. Sathishkumar, M. Vinothkumar, S. Devaraj Varatharaj and S. M. Gowthaman, "Design And Development Of Automatic Fire Detection Using Sms And Voice Alert System", International Journal of Scientific & Engineering Research, vol. 7, no. 5, pp. 114-117, 2016.