Minia University
Faculty of Engineering
Minia, Egypt

Final Year Project Thesis

In preparation for graduation

Department: Department of Electrical Engineering

Safety and security system with FOTA and IOT

Supervised by

Prof./:
Kamel H. Rahouma

Dr./:
Gerges M. Salama

Team Members

Abanoub Kamal Boshra
Asmaa Elsayed Reyad
Adel Magdy Maharous
Efraim Youssef William
Salma Salah ElDin Abdullah
Magdy Edwar Farid

College Year
2020/2021

Abstract

Internet of Things (IoT) conceptualizes the idea of remotely connecting and monitoring real world objects (things) through the Internet. When it comes to our house, this concept can be aptly incorporated to make it smarter, safer and automated. This IoT project focuses on building a smart wireless home security and safety systems which sends alerts to the owner by using Internet in case of any trespass and raises an alarm optionally. Besides, the same can also be utilized for home automation by making use of the same set of sensors. The leverage obtained by preferring this system over the similar kinds of existing systems is that the alerts and the status sent by the WiFi connected micro-controller managed system can be received by the user on his phone from any distance irrespective of whether his mobile phone is connected to the internet. The micro-controller used in the current prototype is the Arm cortex m3 and WiFi Module is esp 8266 shield making use of which all the electrical appliances inside the home can be controlled and managed.

Acknowledgments

We would, first and foremost, like to thank our supervisors Professor Kamel H. Rahoma and Doctor Gerges M. Salama. Their guidance and support throughout our graduation project have been greatly appreciated. They are very insightful in directing research topics and are also willing to listen to our opinions. Through the project work, they always helped us making our work better and innovative. What we have learned under their direction will greatly benefit us in our future and career development.

We would also like to thank Eng. Hany Melad for their great efforts in helping us understand Bootloader system with some sources about Bootloader and FOTA, so that we could apply these algorithms on our embedded system.

On the Embedded part, thanks to Eng. Ahmed Zakaria and Eng.Ahmed Elshamy to help us understand it and get our hands on its techniques, and suggest some solutions to some of the problems we encountered, especially in part of timer peripheral.

As for the IOT part, we are very grateful to Eng. Abdallah Omer in helping us understanding iot with some sources he gave us.

As for the ARM STM32 Course , We appreciate all thanks for IMT school for providing us with the ARM Diploma recorded Lectures of Eng/ Ahmed Assaf that helped us too much throughout the course. and we can not forget NTI that provided us with the place and workspace of CREATIVA to sit and work as a team . really it was wonderful.

Finally, we appreciate the unconditional love and support of our family, especially to our parents for their encouragement and patience, they kept our motivated when things were not going well.

Contents

List of Figures

List of Tables

1

INTRODUCTION

As there are many dangers around people mainly inside the building and outside it like theft and leakage of harmful gases and fires, all of these dangers which may lead to death can't to be faced by persons only, because, human being can't detect the leakage of little of natural gas which may causes suffocation and if it increases and there is flame will cause explosion.

So we chose this idea as a project because it is very applicable and we can protect buildings and people from their surrounding dangers and this is the problem which we aim to overcome. We can subdivide This project according to the Technologies which are used to achieve this idea and according to the main services which can be done.

1.1 The used technologies

We use Embedded Systems, IOT and FOTA Technologies

1.1.1 Embedded Systems

Embedded Systems are present in every aspect of our life. Starting from the portable devices such as mobile phone and the digital hand watch, passing by the home appliances such as television and washing machines until the highest technology vehicles and most complex industrial control systems.

Definition An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, with real-time computing constraints, include hardware, software and mechanical parts

Components of Embedded System Each Embedded System consists of Micro-controllers, which are microcomputers incorporating the processor, RAM, ROM and I/O ports and some peripherals which make them able to process many types of data and do tasks correctly.

Architecture of Embedded Systems The architecture of an embedded system is an abstraction of the embedded device, meaning that it is a generalization of the system that typically does not show detailed implementation information such as software source code or hardware circuit design. At the architectural level, the hardware and software components in an embedded system are instead represented as some composition of interacting elements. Elements are representations of hardware and/or software whose implementation details have been abstracted out, leaving only behavioral and inter-relationship information. Architectural elements can be internally integrated within the embedded device, or exist externally to the embedded system and interact with internal elements. In short, an embedded architecture includes elements of the embedded system, elements interacting with an embedded system, the properties of each of the individual elements, and the interactive relationships between the elements.

1.1.2 IOT Technology

Definition It is a network of physical objects or people called "things" that are embedded with software, electronics, network, and sensors that allows these objects to collect and exchange data.

IOT makes once "dumb" devices "smarter" by giving them the ability to send data over the internet, allowing the device to communicate with people and other IOT-enabled things

Components of IOT Technology Figure 1.1 shows the main components of IOT System which are: Sensors/Devices; a key component that helps you to collect live data from the surrounding environment. All this data may have various levels of complexities. Connectivity All the collected data is sent to a cloud infrastructure. The sensors should be connected to the cloud using various mediums of communications. These communication mediums include mobile or satellite networks, Bluetooth, WIFI, WAN. Data Processing Once that data is collected, and

Figure 1.1: IOT Network

it gets to the cloud, the software performs processing on the gathered data. User Interface The information needs to be available to the end-user in some way which can be achieved by triggering alarms on their phones or sending them notification through email or text message.

1.1.3 FOTA (Flash Over The Air) Technology

Definition FOTA (Firmware Over The Air) This is a special feature supported by some phones, where users can update their handset firmware over the carrier network. It removes the need of special cables, computers or third-party programs and in our project we can update our subsystems (Fire system – Gas System – Security System) from any area and can flash on the systems any new versions and updates from the software.

Components of FOTA Technology Figure 1.2 shows all the components we will use to apply FOTA principle and concept in our project.

- WIFI Module : and this is the component over which the user can receive the update of the new software of the subsystems.
- Boot-loader : and this will be explained in details in chapter three.

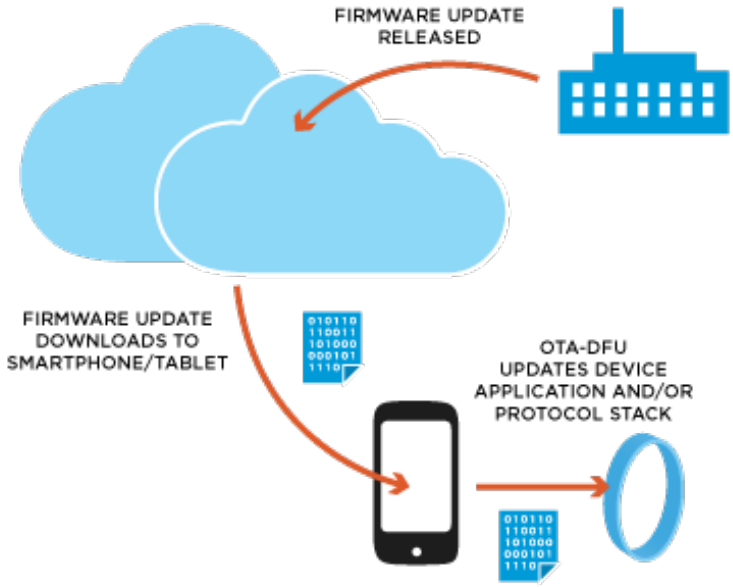

Figure 1.2: FOTA Applications

- User mobile : This is the tool that the user can control anything in his systems from anywhere and can be up-to-date with every new software, the mobile will search for updates or the system provider will let him know (with a notification on the mobile) that there is update. And he has the freedom to download it or not.

1.2 Services provided by the project

Mainly our project is subdivided into three manners:

- **Safety from Harmful Gases** which focus on the dangers of leakage some gases Like the Natural Gas; which is leaked from sources of gas inside the building like Ovens and Stoves, and another harmful gas is CO gas which may be leaked from Heater.
- **Safety From Fires** Which results from burning some things in the building.
- **Security From thieves** Because thieves can enter to building from external ports of the house or by copying the key of the outdoor.

Because we have three different Services we have three Subsystems: Safety from

harmful gases, Safety from fires and Security Systems. And we will use the previous technologies to achieve the most requirements of the user which may needed.

1. For IOT Technology:

   We will use ESP8266 Microcontroller in station mode as a WIFI module connected with access point to manage Subsystems.

2. For FOTA Technology:

   We will use WIFI Module and Boot-loader to achieve this purpose and to control our systems from in place in the world.

Every subsystem will have a microcontroller (STM32) and a Wi-Fi Module

1. For Security System:

   - There is a keypad at the outside doors and asks anyone who want to enter the building about the passcode (which is defined previously by the owner) and in case of entering it wrongly three times the system will unlock the door and will send a message by a Wi-Fi module to the owner to alert him.
   - There are sensors to sense the distance at windows and balconies to prevent thieves from entering to the building as when thief pass from it the system will give a great sound to alert people and will send a message to the owner by a Wi-Fi module to alert him that someone tries to enter his place (house).

2. For Safety From Fires System:

   - There are sensors to detect the smoke gas which results from burning and two flame sensors at the place protected from fires and when these sensors read a defined value which is processed by micro, it will send signals to open the window/balcony using motors and will alert people inside the building and will send a message to the owner by a Wi-Fi module.

- There are sensors surrounding the sources of gas and stove to detect the gases like CO-CO2 and natural gas and when the sensors sense an amount of gas it will send to the micro controller to open the sources of air (Windows) and turning off the source of gas using suitable motors and finally send a message to the owner by a Wi-Fi module.
- The owner can control (turning off) the main source of gas

1.3 Document Structure

We will illustrate in details how each of the previous subsystems works in the following chapters as:

In Chapter 2, We will look at Literary review and will show how other engineers handle these systems in different manners

In Chapter 3, we Will present each subsystem in details including the used sensors and how they are connected to the micro and how the power of ARM which make the systems work efficiently. After that, we will present the technologies used for each subsystem in details.

In Chapter 4, We will show the results of each subsystem due to the used sequence for handling them and due to the used component and technologies. Also, We will present some figures for the connections of subsystems. Finally, We will compare these systems with similar systems which is presented in Chapter 2.

In Chapter 5, we concluded the three Subsystems with the technologies which are used to handle them. and compare the results of these systems with those systems introduced in Chapter 2. Also, Suggestions for developing these systems or the used technologies.

2

THE STATE-OF-THE-ART HOME SAFETY AND SECURITY

2.1 Overview

Security is an important aspect of embedded system design. The characteristics of embedded systems give rise to a number of novel vulnerabilities. A variety of different solutions are being developed to address these security problems. In this paper, we provide a brief overview of important research topics in this domain.

In this chapter we will discuss the literature review and history of our systems, and how they were developed in the past and what they used to achieve their targets and to protect homes and to secure the entrances of the house.

2.2 Introduction

Security in embedded systems is a topic that has received an increasing amount of attention from industry and academia in recent years. Embedded systems are being deployed in a wide range of application areas ranging from control of safety-critical systems to data collection in hostile environments. These devices are inherently vulnerable to many operational problems and intentional attacks due to their embedded nature. Network connectivity opens even more avenues for remote exploits. In response, security solutions are being developed to provide robustness, protection from attacks, and recovery capabilities and we will discuss especially the security and safety related to homes and theft.

2.3 Fire Alarm System Papers

In this part of the chapter we will mention the past and development (progress) of the Fire Alarm System and we will give some scientific papers and researches that talked about this topic and what components they used.

In [9] ZigBee-equipped wireless smoke sensors along with IT technology can activate alarms if are occurs. In this study, the ubiquitous technology is implemented for the purpose of ensuring are safety in residential buildings through wireless are

detection and extinguishing system.

His system also aimed at reducing installation cost due to space restriction and promoting ease and practicality of installation in existing residential buildings. Unlike wired interconnection which may only be practical for use in new construction, especially if the wires are laid out without cutting walls and ceilings (or doors in multistory residence), the wireless interconnected sensors can be conveniently retrotted in buildings without costly wire installations. In effect, the building interior design is not compromised and total installation cost can be significantly reduced.

The main ideas of the proposed are detection and extinguishing system were as follows: Firstly, in housing condition where it is not easy or impractical to install conventional are detection equipment, a smart are sensor network using wireless communication system that activates and sends warning alarm within the building, to the building residents or owner, and the are department was developed.

Secondly, to implement an improved compact water-based are-extinguishing sprinkler system with higher discharge so that the are can be effectively extinguished at the early stage. The compactness of the extinguishing system is believed to cut the total installation cost. The two systems were integrated for the development of practical wireless are detection and extinguishing system for residential building applications.

Experimental equipment for water supply with specific type of sprinklers and wireless communication were developed and tested to verify the system performance.

In residential structures, a potential fire can be indicated by the presence of smoke, heat, gas, or their combination. The smart fire protection system presented in this paper is initiated upon the detection of these early indicators of fire. Generally, a warning signal or alarm is produced when the concentration of detected smoke, carbon monoxide (CO), liquefied natural gas (LNG), or liquefied petroleum gas (LPG) has met the criteria for alarm activation, that is, above thresh-Old values, These sensors are interfaced with the control system so an automatic fire notification system is executed at Telephone networking is in the control system thereby

automatic notification call or message is sent to the fire department and residence owners who are off the premises the instant the fire alarm is activated. In an unlikely event of fire, the sprinkler system is activated allowing water to flow out of the sprinkler heads to extinguish the fire. Since the residential sprinklers are temperature-activated, only the sprinkler head closest to the fire source is set off, thus, reducing damage to properties. Water pressure information in the sprinkler line is sent into the control system through the standalone type alarm valve. The water pressure required to extinguish the ongoing fire is supplied by a water pump that is also interfaced with the control system. When fire is totally suppressed, the water supply is automatically shut off. In case that LNG or LPG leakage is detected which is attributed by the gas concentration above the preset limits of the sensor, the gas line is automatically shut off and the gas bureau and residence owner are instantaneously notified.

In [4] The designed firefighting system is able to detect and deal with over temperature, smoke and flame. After develop and testing the software the automatic firefighting system has successfully completed the tasks as expected and as outlined in the report. The system is capable of indicating its status on a LED, buzzer, and detecting and putting out fire. The ATmega16 microcontroller is used to process the sensor circuitry input and control the indicator panel, and also used to control a firefighting pump to put out flames when detected. A simulation of the system is created and it gave a good performance.

The automatic firefighting designed system is controlled by a microcontroller unit describe building consist of three rooms each room contain three sensors (temperature, smoke, and flame). The system uses set of LEDs, buzzer to indicate the presence of smoke, over temperature or fire in any room, and a firefighting pump on standby to put out the flames when needed. Figure 2.1 illustrate the hardware design of implemented automatic firefighting system. The system performs automatic fire fighting task when the system assures the fire occurrence. Figure 2.2 below shows the results when operating the system. The figure indicates the action taken

for each happening.

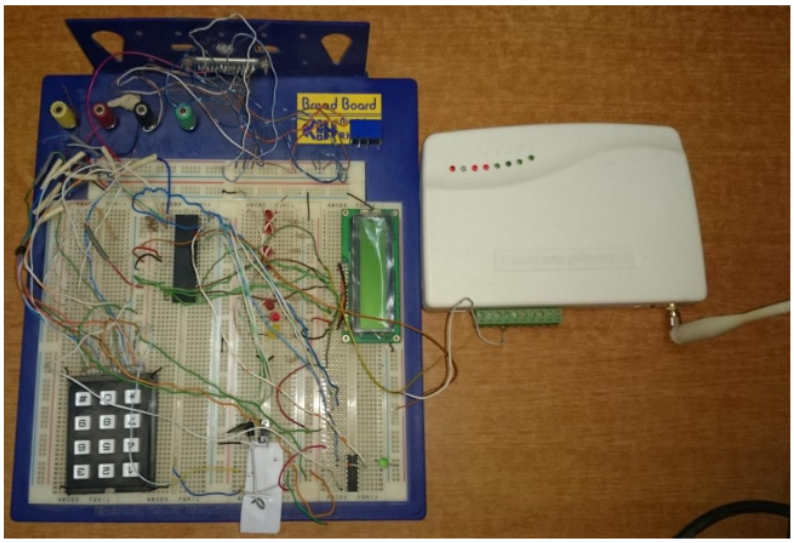

Figure 2.1: STM32 Microcontroller

Temp. sensor	Flame sensor	Smoke sensor	Hazard LED's and buzzer	ACTION
0	0	0	Off	No action
0	0	1	Off	Alarm led on
0	1	0	On	Pump on
0	1	1	On	Pump on
1	0	0	Off	Alarm led on
1	0	1	On	Pump on
1	1	0	On	Pump on
1	1	1	On	Pump on

Figure 2.2: Results when system operates

2.4 Security system Papers

In [1] Based on Figure 2.3, this paper used an ESPresso Lite V2 as the microcontroller. The ESPresso Lite V2 will be connected to the UC00A that will supplied a 3.3V to power up the board. As the ESPresso Lite V2 operated, it will auto-

matically connected to the internet via build in Wi-Fi board. This microcontroller used ESP-WROOM02 as the WiFi board and send data to the FAVORIOT platform. The inputs for this project are the data from the PIR sensor and IR sensor which will monitor the home surrounding and send the data to the ESPresso Lite V2. This project will only operate when the user are not at home, thus, a 'Blynk' application will be used to act as a switch to turn the project 'ON' and 'OFF'. The OLED display will show the output for each conditions and the data is send to the webserver which is FAVORIOT. The webserver will automatically send an alert to the user via smartphone.

Security of connection between microcontroller and mobile phone is encrypted providing a secure system that is not vulnerable to hijacking. Secure connection can be provided as the password need to be hardcoded via direct connection.

When the product has been configured properly, the product can be mounted at the area where user wanted to monitor. Whenever the system is activated by click on Blynk application, any motion and door activity will be recorded and sent to Expresso Lite V2.0 microcontroller. Any activity can be seen in FAVORIOT application, as well as wireless notification in order to use IoT technology.

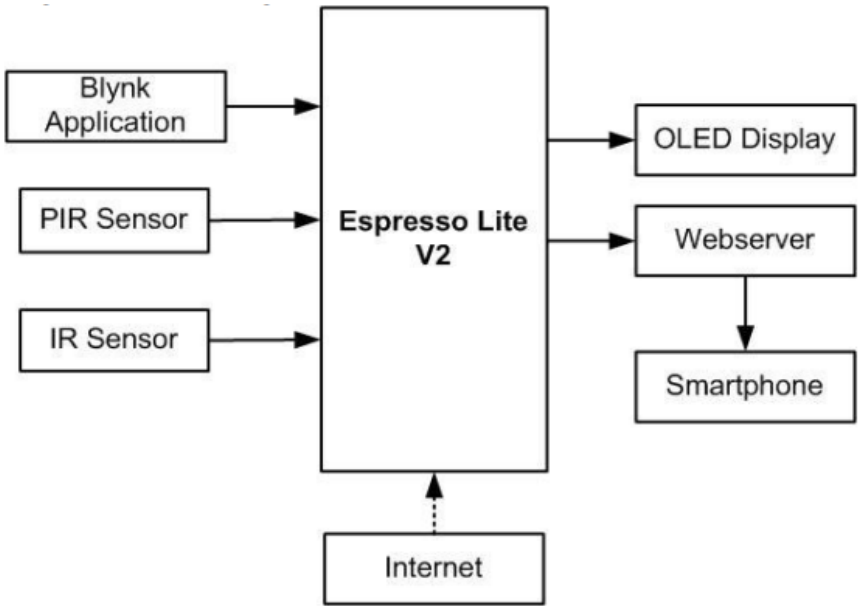

Figure 2.3: The block diagram of project

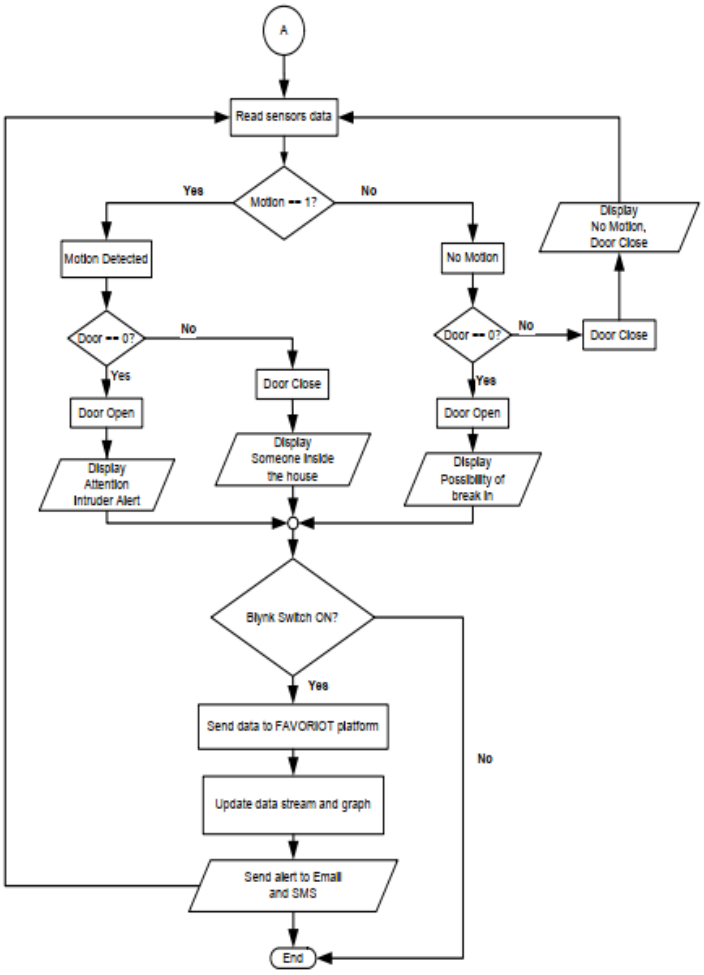

Figure 2.4: Flaw chart of project

System in Figures 2.5 and 2.6 proposed by Isa and Sklavos [5]

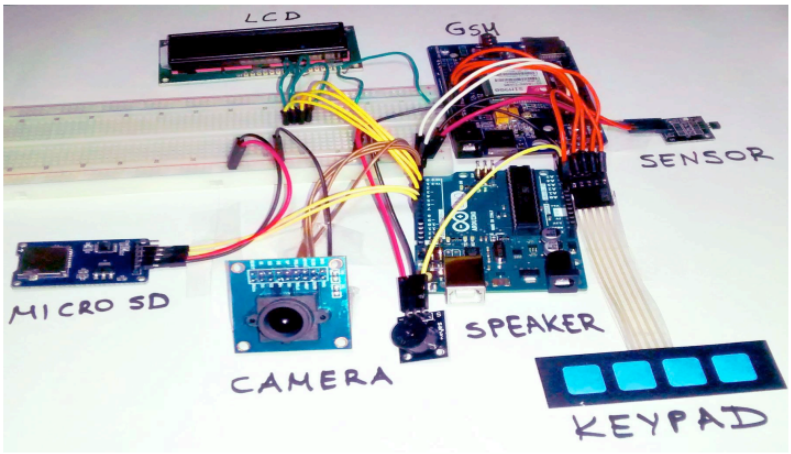

Figure 2.5: Circuit of project

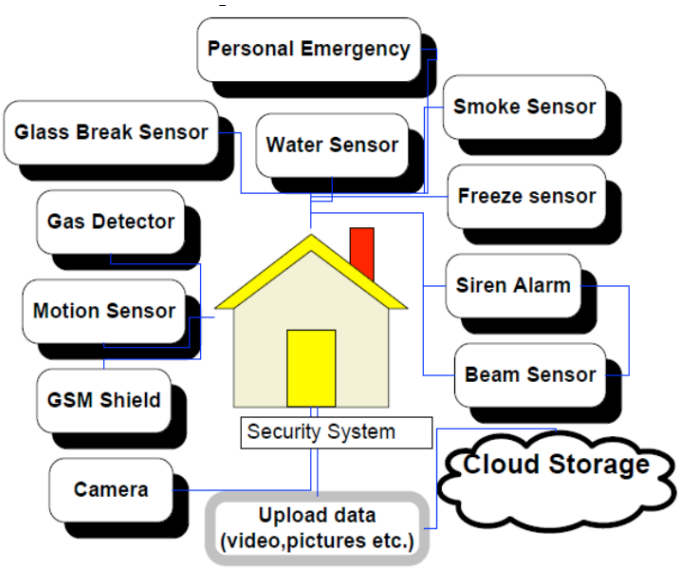

Figure 2.6: Components of project

2.5 Gas protection papers

In [7] When the LPG gets to the base weight, the GSM sends the message to the proprietor by the assistance of Arduino. is accustomed to sending the message to the client with the assistance of GSM. The LCD shows the heaviness of the chamber consistently. This program is created to send the message through GSM to the given number of the client. The gas sensor MQ6 to detect the gas around the place, after that microcontroller automatically switch ON the signal and sends the message to the proprietor The message sent to the fixed number automatically and access remotely with the assistance of GSM. At the point when the heaviness of the gas chamber gets least; GSM sends the message to the proprietor and cautions the proprietor about the leakage of the gas, see Figure 2.7.

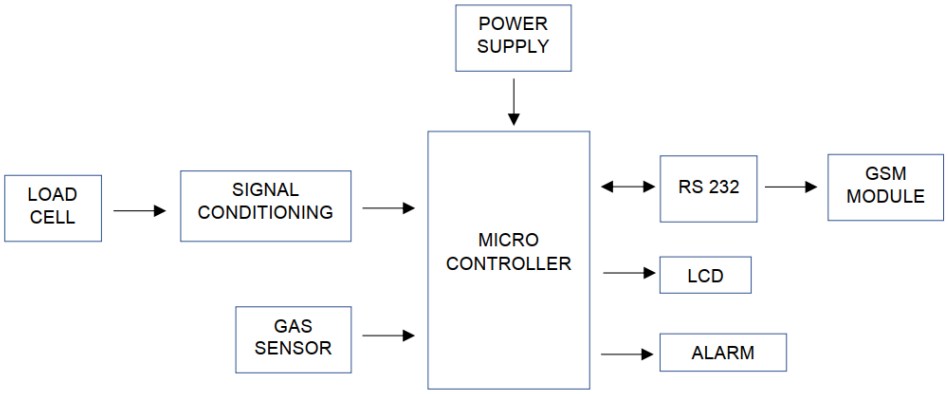

Figure 2.7: Flaw chart of circuits

The main aim of this system is to estimate the GAS level and leakage of the gas from the room or an industry. The leakage of the gas will be estimated with the help of IGLS system using MQ6 gas sensor. This gas sensor senses the gas and sends the message to the user with the help of GSM module. Liquefied Petroleum Gas (LPG) sensor is simple and easy to detect the gas level. Gas sensor has fast response and high sensitivity module. Liquefied Petroleum Gas (LPG) sensor is simple and easy to detect the gas level. Gas sensor has fast response and high sensitivity.

In this paper [6], semiconductor sensors are used to detect LPG gas. An MQ6 semiconductor sensor is used. Sensitive material of the MQ-6 gas sensor is SnO2, which has lower conductivity in clean air. When the target combustible gas exists, the sensor conductivity increases along with the rising gas concentration. The MQ6 gas sensor has a high sensitivity to Propane, Butane and LPG, and response to Natural gas. The sensor could be used to detect different combustible gasses, especially Methane; it hasa lowcost and is suitable for different applications. The MQ-6 can detect gas concentrations anywhere from 200 to 10,000 ppm. The sensor's output is an analog resistance. Figure 2.8 shows the block diagram of the gas leakage detection and alert system.

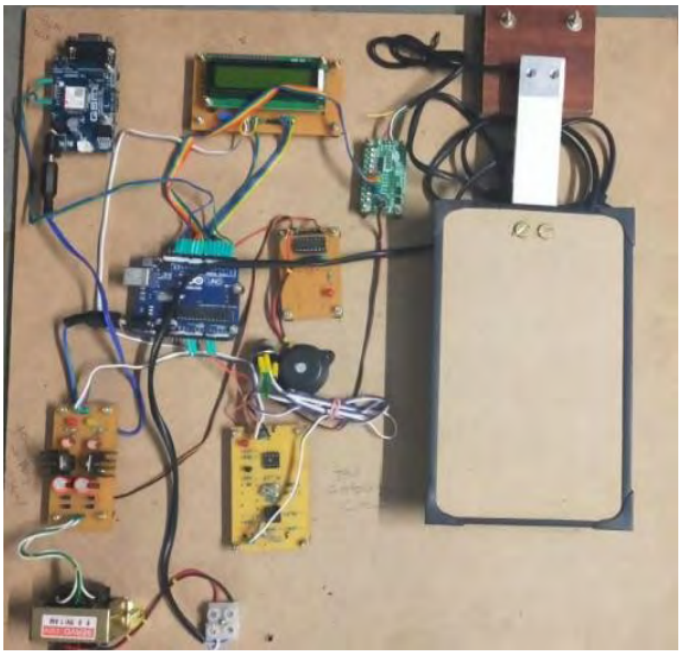

Figure 2.8: Circuit

Figure 2.9 shows the circuit diagram that was designed using Proteus libraries. This system is based on Arduino UNO R3 and MQ-6 gas sensor. When the sensor detects gas in atmosphere, it will give a digital output of 1 and if gas is not detected the sensor will give a digital output of 0. Arduino will take the sensor output as the digital input. If sensor output is high, then the buzzer will start tuning and the LCD will show that "Gas detected: Yes". If sensor output is low then the buzzer will not be tuning, LCD will show that "Gas detected: No". The detector incorporates a MQ-6 sensor (with gas detection range of 300–10,000 ppm) as the LPG gas sensor, PIC16F690 microcontroller as the control unit, LCD for displaying gas concentration, a buzzer as an alarm and a number of LEDs to indicate the gas leakage status. The microcontroller senses the presence of a gas when the voltages signal from the MQ-6 sensor goes beyond a certain level and gives an audiovisual alarm.

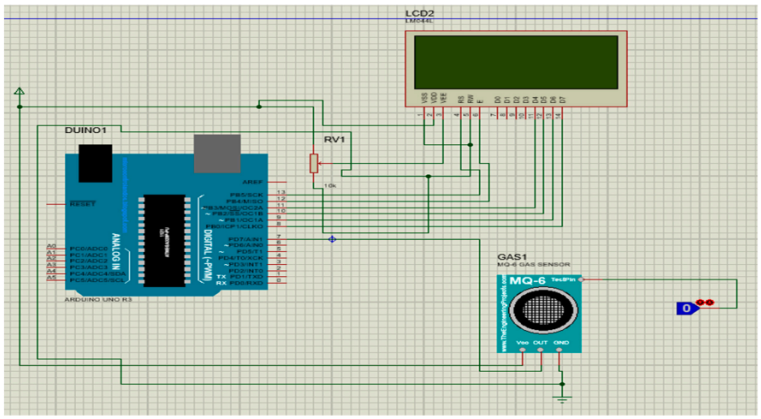

Figure 2.9: Simulation of circuit

Figures 2.10 and 2.11 represent a system proposed by Raj et al. [10].

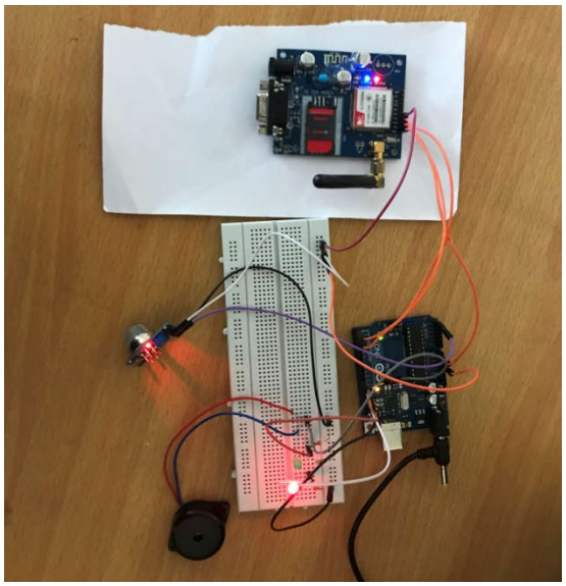

Figure 2.10: Circuit1

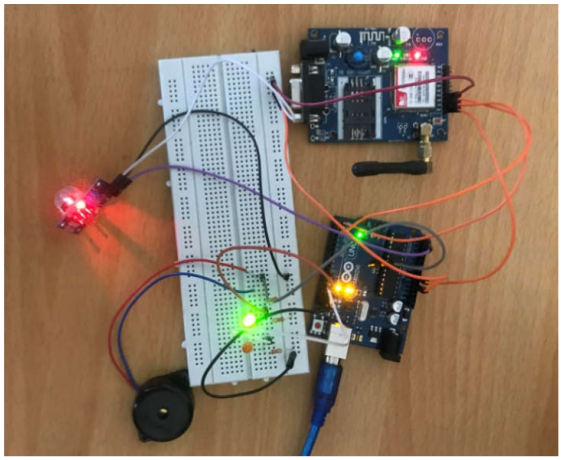

Figure 2.11: Circuit2

3

METHODOLOGY & THE SUBSYSTEMS

3.1 Introduction

Incidents like thefts, fire and LPG gas leakage are very common these days. What is un common, is people's awareness about different systems like a fire detector, gas leakage detectors, etc. Installing all these different detectors in order to keep the house secured is also something that is difficult to maintain. Here, we have designed an integrated home security system that would help people secure their houses from such incidents. IOT and Arm cortex m3 microcontroller based Home safety and Security System project is designed to help an individual secure his/her house from theft, fire and LPG gas leakage – all in one. This project uses four different sensors, from which data is sent over a website through IOT. Internet of Things (IoT) is basically, the network of 'things' by which physical things can exchange data with the help of sensors, electronics, software, and connectivity. These systems do not require any human interaction, and ability to update the system using (FOTA).

IOT and Arm cortex M3 Based Safety and Security System uses four Sensors, namely, Temperature, Flame, LPG and IR sensors. Data from these sensors is then sent to the Arduino, which has an inbuilt signal converter. Microcontroller then sends data over to the Wi-Fi module – ESP8266. ESP8266 is a chip used for connecting micro-controllers to the Wi-Fi network and make TCP/IP connections and send data. Data, which is sensed by these sensors, is then sent to the IOT. To elaborate on the theft detection, we have connected a password module by which a user can enter the password. The door would open only if the password entered is correct. . If somebody tries to enter wrong password for consecutively 3 times. message will be send to the user and the user turn the door off by IOT .To demonstrate the door, we have used a DC motor. The ultrasonic sensor needs to be installed on the windows, which activate the buzzer if someone entre the window, The buzzer will turn ON even when it detects incense flame, and send a message to the user if there is a gas leakage and turn the gas source off, we can update the System using boat loader and IOT server (FOTA).

3.2 The proposed system block diagram

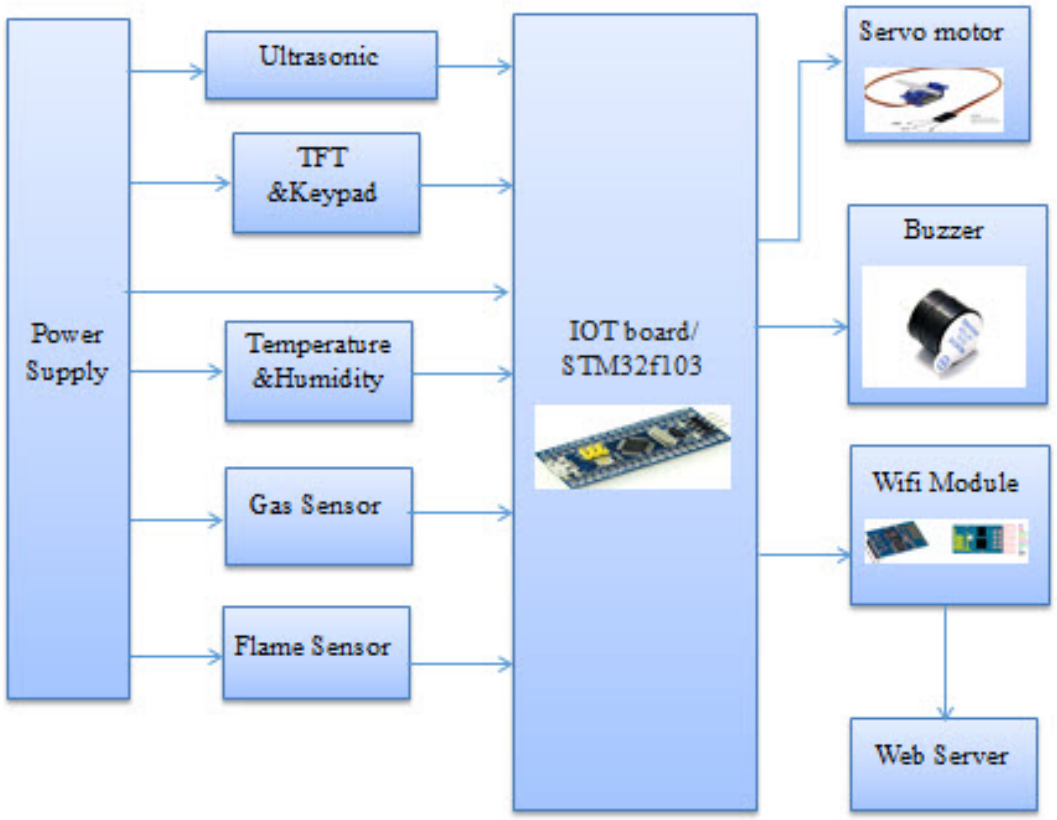

Figure 3.1: The proposed system block diagram.

3.3 The advantages of the proposed system

- Helps Protect Your Belongings

  Integrating a smart home security system into your house can better protect your belongings and valuables. In fact, recent studies have shown that burglaries cost an average of $1,700 per victim.

  Since most burglars seek out expensive electronics, drugs, and jewelry, a smart home security system can help alert you and authorities if you are burgled. This can increase the chances of your burglar getting caught and lowering the cost of your home being ransacked.

- Integrates Well With Home Automation

One of the key features of owning a home security system is it integrating well with your other smart home devices. This includes smart lights, door locks, and even smart thermostats. Everything can be easily connected and accessed via your smart home security system website or app on your computer, smartphone, or tablet. You are even allowed the option to set up modes for your home for when you get home. For example, you can set a mode for when you get home from work to turn on your TV, turn on the AC, or even activate Alexa to give you the latest news. These simple conveniences can help you live more efficiently and happier.

- You Can Access It From Virtually Anywhere

  Since your smart home security system can connect to your laptop, smartphone, or tablet, you can access it from home or on-the-go. Whether you need to view your security cameras, lock your doors, or adjust your house's thermostat, there's no limit what you can do from anywhere in the world. Plus, the app and website platform you use for your smart home security system will have added protection and security for your benefit.

- You Will Be Protected 24/7

  Some conventional home security systems require you to turn it on or off every time you arrive or leave your home. Instead, your smart home security system is on 24/7 and can send you alerts throughout the day based on the way you program it. This can help you prevent forgetting to turn it back on and leaving your house unprotected. Plus, some systems even allow motion sensing and can trigger emergency notifications if it suspects something is wrong.

- It's Never Been Simpler To Use

  New smart home security systems are going wireless, which means you can carry them with you whenever you like. This also makes moving to a new home much easier. Your wireless sensors and cameras are simple to setup and

can be up and running within minutes. Plus, you can do everything through your smart home app without the need for even WIFI.

- It's A Complex System

  Smart home security systems are much more complex and intuitive systems than conventional security methods. Instead of simply tracking main entry points, a smart home system can track actual behaviors and alert you via text messages if they suspect any suspicious movement, For example, your new smart home security system can detect and alert you if a burglar cut a phone line. Especially, since this is a common behavior to do before they rob a home. Additionally, your smart home system connects via an app or desktop platform that can send you alerts if it ever loses connection when protecting your home.

- Prevent Theft

  The best way to prevent theft or damage to your home is to prevent burglars from targeting your home in the first place. But how exactly do you do that. With the use of smart home security systems, The mere existence of one of these alarms is often enough to keep robbers out of a home.

  In fact, a survey completed in 2012 found that 60% of theft offenders would skip a house if it had a security system. They instead moved on to easier targets, i.e., Home's without security systems. Plus, of those who realized the home had an alarm while they robbing it, 50 percent gave up on their attempted crime.

- Lower Your Home Insurance Rates

  Who doesn't love saving money? A lot of insurance companies offer discounts or reduced premiums for homeowners who have security systems in their homes. This is because homes with these systems have a reduced chance of experiencing loss, fire, and water damage. This means that the insurance company itself will have to process fewer claims, which they reward.

- Save Money

  Did you know that you can actually save money by using smart home technology? It's true! As we mentioned above, simply having a security system in place can save you thousands on prevented robberies and property damage.

  However, a smart home system can help you save money on your electric bill as well. These systems often use less energy than traditional home security systems. Also, if you opt in for full smart home automation, you will see even more savings. It allows you to choose when to turn on and off your lights and what temperature you want your house to be. Not only can this control save you money, but it can also be used to deter thieves. Robbers are less likely to target a house if it looks like someone is at home, so turn on those lights.

- Greater Peace Of Mind

  You may be surprised to discover just how much peace of mind installing a smart home security system will bring. The convenience and protection that this type of system brings are priceless. You will never need to your about the safety of your family, your pets, or your belongings again. Instead, you can relax knowing that your home is safe and secure against all sorts of damage, whether it is caused by humans or an accident, While there are many advantages of installing a home security system, there are even more that accompany using a smart home system. With the added protection, convenience, and ease of use, you will not be disappointed.

3.4 Security Subsystem

3.4.1 Introduction

All over the world, security has been a major concern in every home. Automated security systems are a useful addition to today's home where safety is an important issue. Vision-based security systems have the advantage of being easy to set up,

inexpensive and no obtrusive.

Security of the person is a basic entitlement guaranteed by the Universal Declaration of Human Rights, adopted by the United Nations in 1948. It is also a human right explicitly defined and guaranteed by the European Convention on Human Rights, the Constitution of Canada, the Constitution of South Africa and other laws around the world.

Door and window sensors are also implemented in most home security systems. One part of the system is installed on the door or window itself while the other part is installed on the frame of the door or window. The two part system connects securely when a door or window is closed, creating a security circuit.

The door will opened by servomotor with a lock coupled in its shaft. When wrong password is pressed, error text is displayed in the TFT.

Window sensors use a range of technologies like beam sensors, motion sensors.

3.4.2 Block diagram of security subsystem

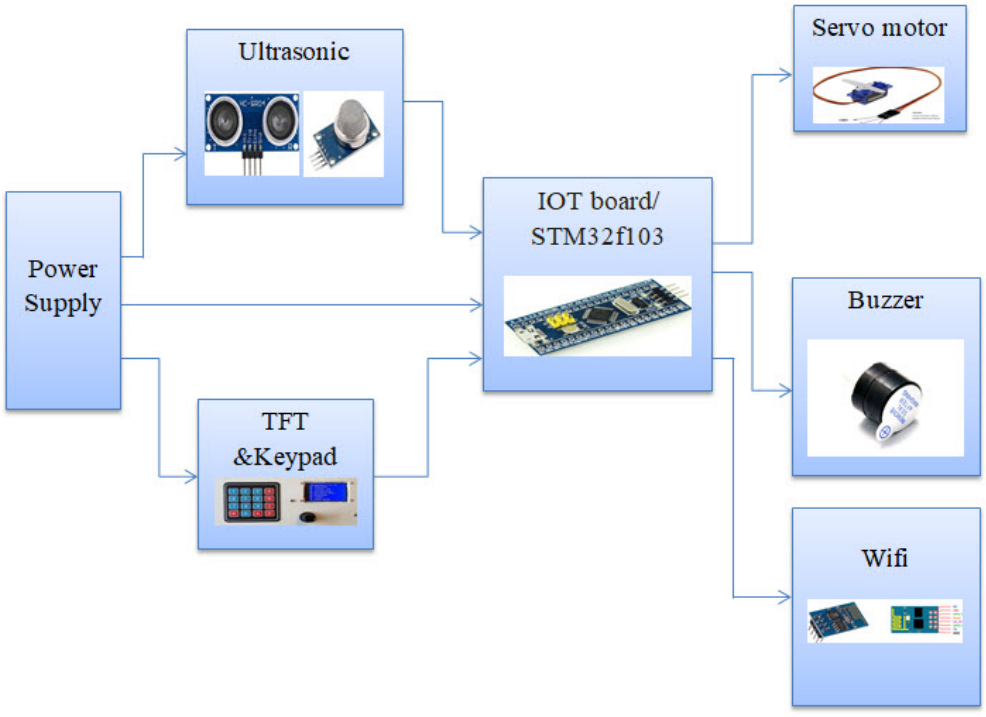

Figure 3.2: Block diagram security subsystem

3.4.2.1 STM32 micro-controller

Our category is Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes

We use many peripheral in ARM like:

- GPIO is input/output (I/O) peripherals to connect small display, buttons, motors, sensors, etc. To "control" a microcontroller, you can put programs onto it and run them. As we will see later
- RCC : to connect clock on used peripheral
- STD TYPES: to definition types of variable
- Timer
- NVIC
- SPI
- Systick
- USART

3.4.2.2 Ultrasonic Ranging module (HC-SR04)

As shown above the HC-SR04 Ultrasonic (US) sensor is a 4 pin module, whose pin names are Vcc, Trigger, Echo and Ground respectively. This sensor is a very popular sensor used in many applications where measuring distance or sensing objects are required. The module has two eyes like projects in the front which forms the Ultrasonic transmitter and Receiver. The sensor works with the simple high school formula of the relation between distance, speed and time.

Figure 3.3: Ultrasonic sensor pin configuration

The Ultrasonic transmitter transmits an ultrasonic wave, this wave travels in air and when it gets objected by any material it gets reflected back toward the sensor this reflected wave is observed by the Ultrasonic receiver module as shown in the picture below

Figure 3.4: Measuring distances technique

Now, to calculate the distance using the above formulae, we should know the Speed and time. Since we are using the Ultrasonic wave we know the universal speed of Ultrasonic wave at room conditions which is 330m/s. The circuitry inbuilt on the module will calculate the time taken for the Ultrasonic wave to come back and turns on the echo pin high for that same particular amount of time, this way we can also know the time taken. Now simply calculate the distance using a microcontroller or microprocessor.

3.4.2.3 TFT Display

As a display device TFT stands for Thin Film Transistor and is used to enhance the operation and usefulness of LCD displays. ... Other benefits of these thin film transistors are they allow for thinner display designs and different pixel designs and arrangements to vastly improve display viewing angles. millions of high-contrast, clear and bright color pixels. TFTs are used in HDTV sets, computer monitors, laptop monitors, tablets, personal media players, smartphones and even feat user phones.

Figure 3.5: TFT Display

3.4.2.4 Keypad 4X4

The 4 x 4 matrix keypad usually is used as input in a project. It has 16 keys in total, which means the same input values. The 4 x 4 Matrix Keypad Module is a non-encoded matrix keypad consisting of 16 keys in parallel. The keys of each row and column are connected through the pins outside – pin R1-R4 as labeled beside control the rows, when L1-L4, the columns, A 4×4 matrix keypad consisting of micro switch buttons. The module has four holes 3mm (M3) holes for mounting.

The pin designations on each PCB are shown on each PCB. Four pins are thus used as an 'x' coordinate and the other 4 as a 'y' coordinate, The module is already equipped with a soldered pin-headers.

Figure 3.6: keypad used

How it work: First test whether any key is pressed down. Connect power to rows, so they are High level. Then set all the rows R1-R4 as Low and then detect the status of the columns. Any column of Low indicates there is key pressing and that the key is among the 4 keys of the column. If all columns are High, it means no key is pressed down. Next, locate the key. Since the column in which the pressed key lies is identified, knowing the line would finalize the testing. Thus, set the rows as Low in turns until any is unveiled accordingly – other rows will still be High. Now the row can be identified. Detect the status of each column in turns. The column tested Low is the one intersecting with the line – their cross point is just the key pressed.

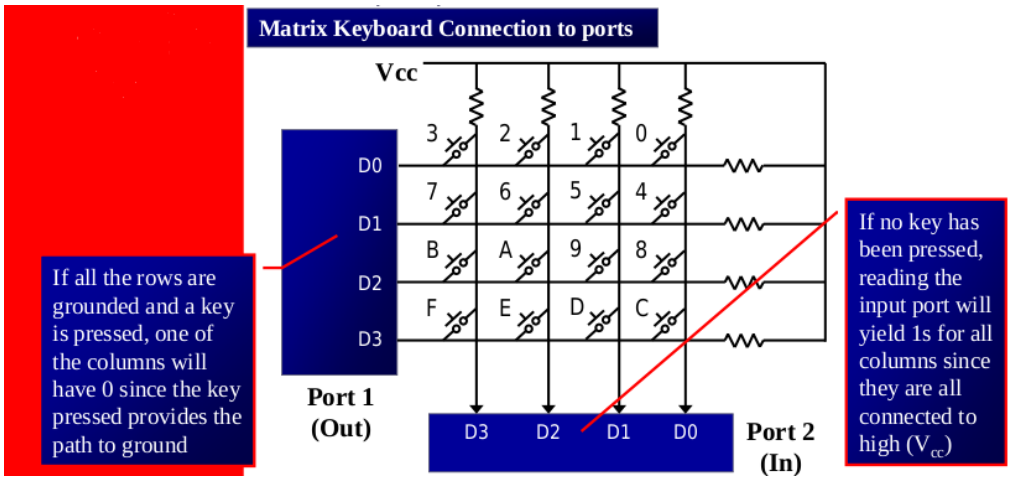

Figure 3.7: how pin connected

3.4.2.5 Servo motor

The servo motor is a closed-loop mechanism that incorporates positional feedback in order to control the rotational or linear speed and position. The motor is controlled with an electric signal, either analog or digital, which determines the amount of movement which represents the final command position for the shaft.

servo motors get their name from the fact that they can be relied upon to operate "exactly as commanded". Any electric motor capable of controlling parameters like position and speed is called a servo motor, regardless of how this control is achieved.

Figure 3.8: servo motor

3.4.2.6 Buzzer

Buzzers can be both fun and useful in electric circuits, We'll use them a lot in Make Create projects, so let's take a look at what is going on inside a buzzer to produce sound.

The buzzer consists of an outside case with two pins to attach it to power and ground. Inside is a piezo element, which consists of a central ceramic disc surrounded by a metal (often bronze) vibration disc.

When current is applied to the buzzer it causes the ceramic disk to contract or expand. Changing the This then causes the surrounding disc to vibrate. That's the sound that you hear. By changing the frequency of the buzzer, the speed of the vibrations changes, which changes the pitch of the resulting sound

Figure 3.9: Buzzer used

3.4.2.7 ESP8266

ESP8266 is a wifi module act as wifi connect with access point in the home, receiving data from S TM32 and sending it to middle area

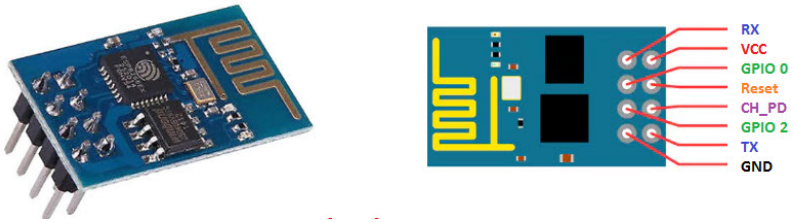

Figure 3.10: ESP8266 (WIFI MODULE)

3.4.3 Circuit Diagram

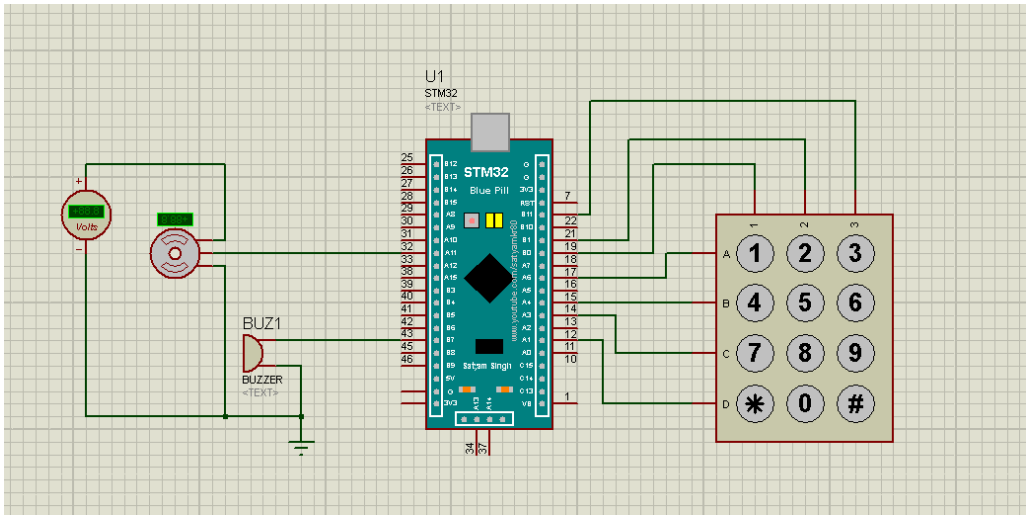

Figure 3.11: Circuit diagram of security subsystem

3.4.4 Algorithm of security

1. Initialize keypad
2. Initialize TFT
3. Initialize Ultrasonic
4. Initialize servo
5. Connected servo and TFT and buzzer and keypad to micro
6. Enter your password by keypad
7. If password write open door by servo
8. If password wrong turn on alarm by buzzer and send massage
9. Connect to wifi using SSID and Password
10. Check if web want to door open or lock
11. Make door open if web send door open
12. Make door closed if web send door lock
13. Check if any thief want to enter your home by ultrasonic and turn on alarm by buzzer and send massage.

3.5 Fire Protection Subsystem

3.5.1 Introduction

We will handle all the previous ideas by separating them into three main Subsystems: Fire, Harmful gases, and Security system.

Every Subsystem include three main branches/Systems:

- Embedded System: to control every sensor and components of the system, and the main component is (STM32F103C8T6) Microcontroller and in this section we will illustrate the reason for choosing this micro.
- IOT System: to interface with user and allow him to take some actions and to send him messages if there is need.
- FOTA Flash Over The Air: to update the whole software of the system when we need to update and add new features.

Our category is Medium-density devices and they are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

The STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 processor is a high performance 32-bit processor designed for the microcontroller market.

It offers significant benefits to developers, including:

- Outstanding processing performance combined with a fast interrupt handling
- Enhanced system debug with extensive breakpoint and trace capabilities
- Efficient processor core, system and memories
- Ultra-low-power consumption with integrated sleep modes
- Platform security

Here our aim is to protect the user and building from fires and as the fire causes some gases (CO2 and smoke gas); we will use Smoke Gas Sensor "MQ_2" to detect it and a flaming sensors to make sure that there is fire and when they sense a discrete value (On Fire value) which is known for the micro-controller (after a calibration process) it will take some procedures.

As we mentioned before we need sensors to detect values of (Smoke gas and flames results from fires) and these values are analog, hence we need "Analog To Digital Converter" peripheral; and STM32 has 2 ADCs (ADC1 ADC2)

From data sheet we can write the driver of this peripheral

3.5.1.1 Block Diagram

To show the process of this System, we use the following block diagram:

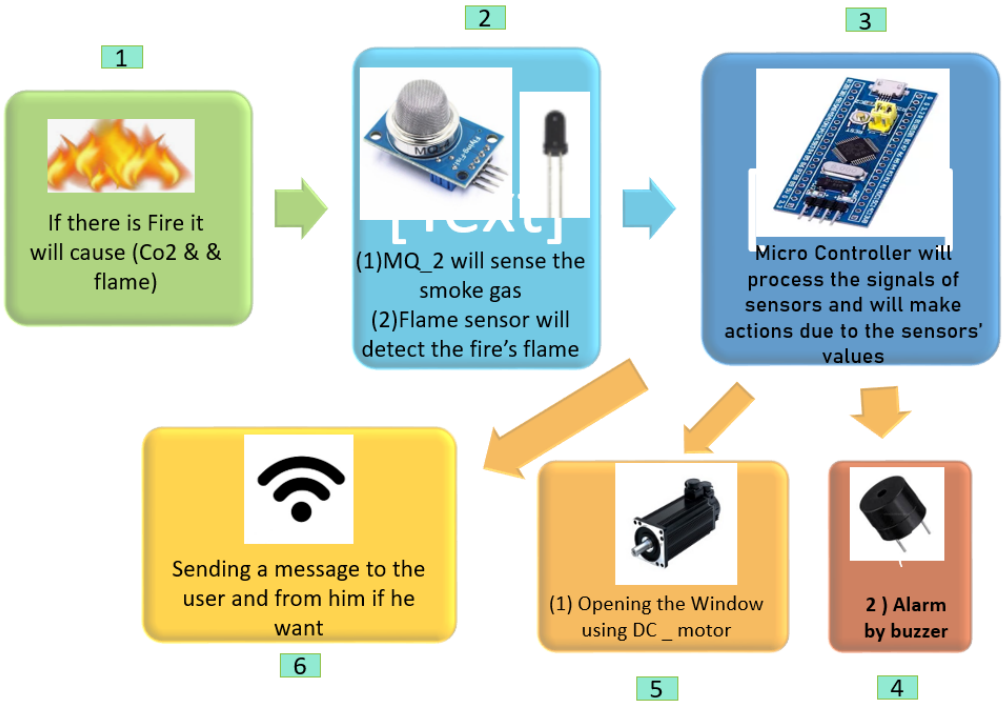

Figure 3.12: Block diagram of Fire subsystem

MQ2 Smoke Sensor or CO2 gas sensor is suitable for detection of Hydrogen, methane and LPG related gases. In short, it can detect combustible gases. It has a wide operating detection range. It has very high sensitivity and very fast response rate. It can even be used in harsh environments. So we can easily use it detect the presence of gas or smoke in our surroundings.

It also has many features which made it suitable for our System:

- Operating Voltage is VCC = ((3:5) V)" DC"
- MQ-2 gas sensor has high sensitivity to propane and smoke, also can detect the natural gas and other flammable steam well. It is with low cost and suitable for different applications of detecting kinds of flammable gases
- Analog output voltage: (0V to VCC V) "Which is used for our system"
- Digital Output Voltage: 0V or 3.3V "VCC" (TTL Logic)

- Preheat duration 20 seconds
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

Flame Sensor Which is used to detect the intensity of surrounding flame; as it is a photo transistor.

It also has many features which made it suitable for our System:

- Detects a flame or a light source of a wavelength in the range of “760nm-1100 nm."
- Detection range: up to 50 cm.
- Adjustable detection range.
- Detection angle about 60 degrees, it is sensitive to the flame spectrum.
- Comparator chip LM393 makes module readings stable.
- Operating voltage 3.3V-5V.
- Analog Output (When the light intensity increases the Output increases)

STM32F103C8T6” Micro Controller As we mentioned before we will use ARM Cortex M3 (STM32F103C8T6) Microcontroller to Process the data collected from sensors and the other components and then take the correct actions corresponding to the algorithm of every subsystem

3.5.1.2 Peripherals Needed for This Sub System From STM32

STM32 CortexM3 contains All the needed peripherals which will be used to handle the Systems; and the peripherals which are used through this system are:

- RCC
- Sys.TicK

Table 3.1: Different clock systems

Clock source	Crystal Oscillator	Ceramic Resonator	RC Oscillator
Accuracy	Medium to High	Medium	Very Low
Cost	High Cost	Lower Cost	Lowest Cost
Setting Time (time between Startup & becoming stable)	Very Low	Medium	High
Temperature immunity	Insensitive to temperature	Insensitive to temperature	Insensitive to temperature
Electromagnetic interface(EMI)	Insensitive to EMI	Insensitive to EMI	Insensitive to EMI
Vibration	Insensitive to Vibration	Insensitive to Vibration	Insensitive to Vibration

- GPIO "General Purpose Input Output".
- ADC "Analog to Digital Converter"

And now we will discuss each of them individually:

RCC: "Reset Clock Control" The optimum clock source for a particular application is determined by a combination of factors including accuracy, cost, power consumption and environmental requirements. The following table summarizes the common oscillator types discussed, together with their strengths and weaknesses.

Figure 3.13: Some of Clock source

Reset and clock control (RCC) peripheral is used to control the clock source of the internal peripherals, as well as the reset signals.

It gets several internal (LSI, HIS and CSI) and external (LSE and HSE) clocks, they are used as clock sources for the hardware blocks.

There are three clock sources can be used to drive the system clock:

1) HSI oscillator clock.

2) HSE oscillator clock.

3) PLL clock .

The devices have the following two secondary clock sources:

- 40 kHz low speed internal RC (LSI RC) which drives the independent watchdog and optionally the RTC used for Auto-wakeup from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal) which optionally drives the real time clock (RTCCLK).

GPIO "General Purpose Input Output" The general purpose I/O ports (GPIO) can be used for driving loads, reading digital and analog signals, controlling external components etc, each of the GPIO ports has two 32-bit configuration registers (GPIO_CRL, GPIO_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR).

Each port bit of GPIOs can be individually configured by software in several modes:

- Input floating.
- Input pull-up.
- Input pull-down.
- Analog.
- Output open-drain.
- Output push-pull.
- Alternate function push-pull.
- Alternate function open-drain
- Alternate function open-drain

Each I/O port bit is freely programmable; it can be configured as 4 types:

- Digital Input.
- Digital Output.

- Analog Input (ADC)/ Output (DAC).
- Alternate function (USART).

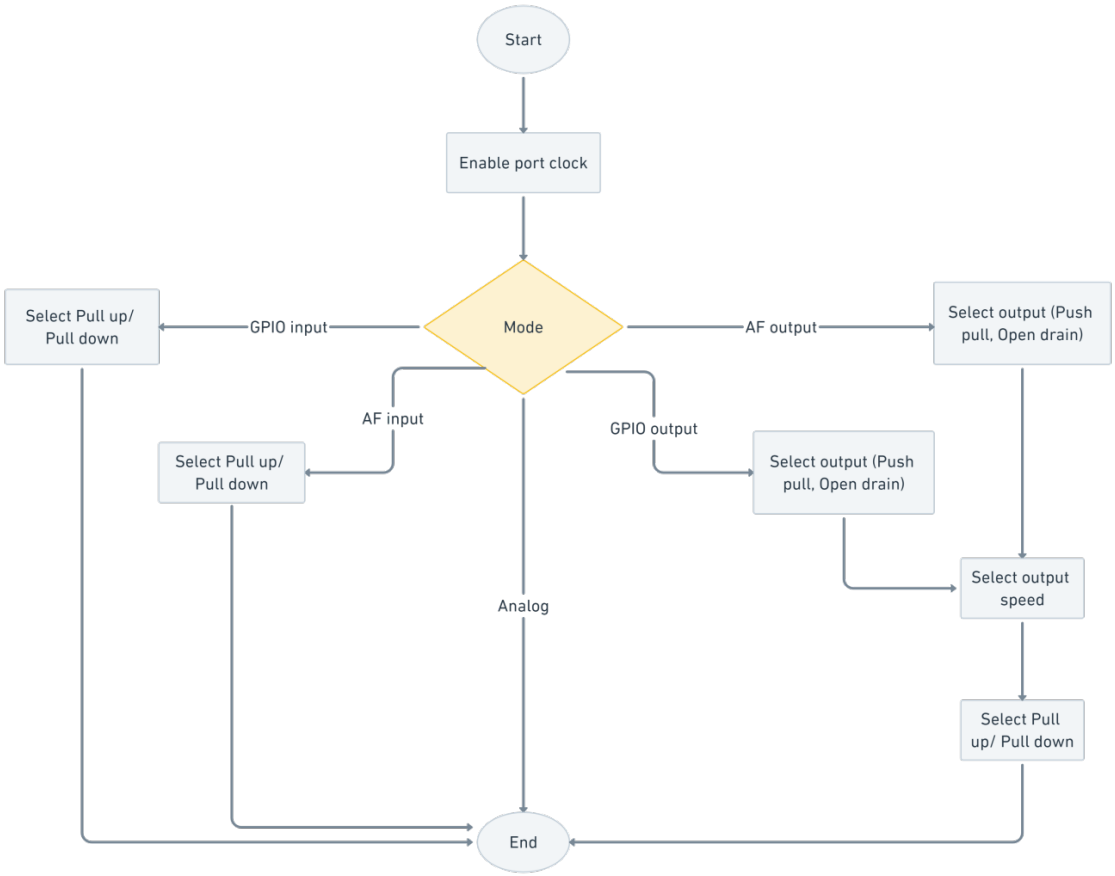

Figure 3.14: GPIO programming flowchart

SysTick The SysTick peripheral is one of the peripherals which are allocated inside ARM cortex-M3 processor. So SysTick is one of the core peripherals.

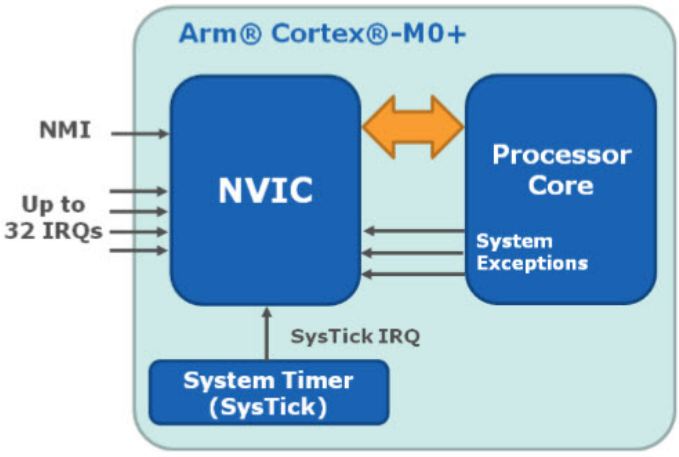

Figure 3.15: ARM Cortex-M0+ processor

SysTick Features

1. SysTick is allocated inside the processor so it will be faster in performing any operations (Core peripheral)
2. Memory Mapped, means that its registers which is allocated inside the core have addresses on the memory, which means that we can access its registers using C language.
3. Access level is privileged so that's why we use SysTick mainly in OS.
4. SysTick is a 24-bit down counter so it means that it can count 224 ticks.
5. SysTick is a down counter timer which means that it can count from (224 1) to zero and it can generate an interrupt when it reaches zero.
6. The main advantage of counting down is when calculating the preload value, you don't need to subtract the preload value from the maximum value of the timer counter to get the actual preload value, but we will directly assign the preload value in the timer preload register to make the timer start to count from.
7. Working clock could be AHB or AHB/8.
8. SysTick timer stops counting when the processor is halted during debugging. Depending on the design of the microcontroller, the SysTick Timer could also be stopped when the processor enters certain type of sleep modes.
9. SysTick timer is portable between all cortex same family (M3).

SysTick operation sequence

1. We need to choose the Input clock source and the options are AHB or AHB/8
2. We need to enable the peripheral interrupt by setting TICKINT bit in The SysTick control and status register.

3. Starting Address of SysTick is at 0xE000 E010
4. We mainly have two registers used to set the beginning of the counter.
5. Load Register: This register used to set the preload value.
6. Value Register: This register is the counter which is decremented by hardware.
7. When the Value Register reaches zero, it will load the Load Register and then start counting down.
8. If the Value Register is accessed using Software, it will be reset without setting the flag.
9. If the Load Register is loaded with a value, the SysTick will start counting from this value, but at the next underflow.
10. If we want to Set the value written in the Load Register to be loaded immediately to the Value Register, then we need to write the preload value into the Load Register and then put any value into the Value Register.
11. The flag will return 1 when the timer is counted to zero.

SysTick has four register

1. STK_CTRL (SysTick control and status register).
2. STK_LOAD (SysTick reload value register).
3. STK_VAL (SysTick current value register).
4. STK_CALIB (SysTick calibration value register).

ADC "Analog to Digital Converter" Analog-to-digital converters are essential for data acquisition. Digital computers use binary (discrete) values, but in the physical world, everything is analog (continuous). Temperature, pressure, humidity, and velocity are a few examples of physical quantities that we deal with every day.

A physical quantity is converted to electrical (voltage, current) signals using a device called a transducer. Transducers used to generate electrical outputs are also referred to as sensors

Some Aspects for ADC Resolution : The ADC has an n-bit resolution, where n can be 8, 10, 12, 16, or even 24 bits. Higher-resolution ADCs provide a smaller step size, where step size is the smallest change that can be discerned by an ADC.

Reference Voltage (VREF+): The maximum voltage can be converted using ADC.

Conversion time: the time it takes the ADC to convert the analog input to a digital number. The conversion time is dictated by the clock source connected to the ADC in addition to the method used for data conversion and technology used in the fabrication of the ADC; Generally, the higher the resolution, the longer it takes to do a conversion.

Start conversion and end-of-conversion signals: For the conversion to be controlled by the CPU, there are needs for start conversion (SC) and end of-conversion (EOC) signals. When SC is activated, the ADC starts converting the analog input value of Vin to a digital number. The amount of time it takes to convert varies depending on the conversion method, clock speed, and resolution. When the data conversion is complete, the end-of-conversion signal notifies the CPU that the converted data is ready to be picked up.

ADC of Cortex M3 Medium Density STM32: There are two successive approximations analog-to-digital converter. They have up to 18 multiplexed channels allowing it measure signals from sixteen external and two internal sources ;(but our micro has 10 channels only).

Main Features of ADC peripheral in STM32:

- 12-bit resolution: hence the converted analog data is stored in 12 bits.
- Interrupt generation at End of Conversion, End of Injected conversion and Analog watchdog event

- Single and continuous conversion; Here we use Single mode.
- Scan mode for automatic conversion of channel 0 to channel 'n'
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual mode (on devices with 2 ADCs or more)
- ADC supply requirement: 2.4 V to 3.6 V
- DMA request generation during regular channel conversion

ADC Functional Description To understand how ADC can work we should look at the block diagram of it:

Channel Selection: There are 16 multiplexed channels.

It is possible to organize the conversions in two groups: regular and injected. A group consists of a sequence of conversions which can be done on any channel and in any order

ADC clock: Like other peripheral modules, the ADC needs a clock to drive the conversion. The ADC clock is enabled by bits of RCC_APB2ENR (RCC APB2 peripheral clock enable register) register in the Reset and Control (RCC) section of the CPU. The conversion clock is derived from the SYSCLK.

Single conversion mode: In Single conversion mode the ADC does one conversion.

Analog watchdog: The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a low threshold or above a high threshold. These thresholds are programmed in the 12 least significant bits of the ADC_HTR

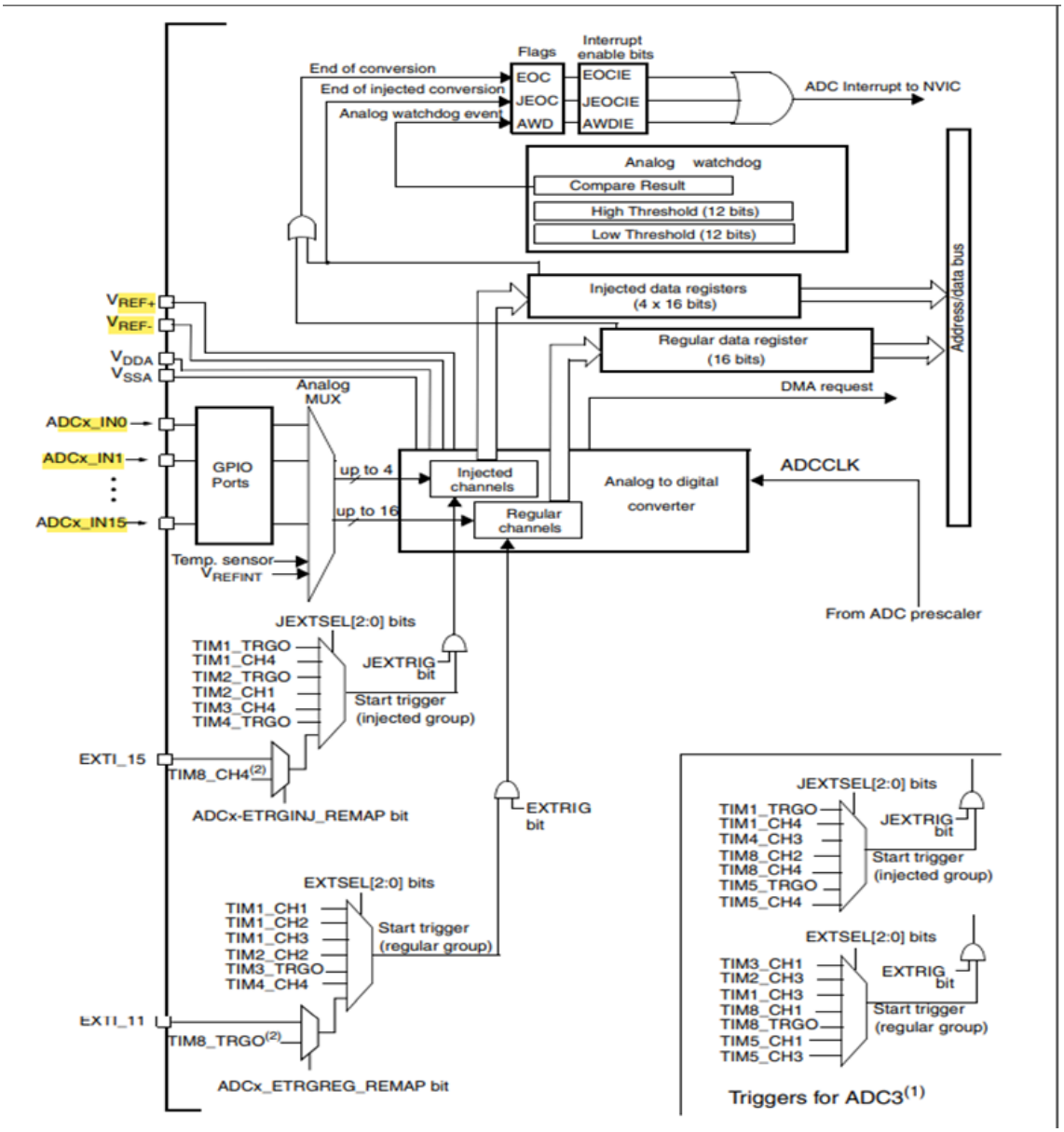

Figure 3.16: ADC Diagram

and ADC_LTR 16-bit registers. An interrupt can be enabled by using the AWDIE bit in the ADC_CR1 register

Data alignment: As there are 16 bit for the register of Aligning data, it selects the alignment of data stored after conversion. Data can be left or right aligned:

If we choose right alignment: the data will be stored in least significant bits.

If we choose left alignment: the data will be stored in most significant bits.

Channel-by-channel programmable sample time: ADC samples the input voltage for a number of ADC_CLK cycles which can be modified. Each channel can be sampled with a different sample time. The total conversion time is calculated as follows: T_conv = Sampling time + 12.5 cycle

ADC interrupts: An interrupt can be produced on end of conversion for regular and injected groups and when the analog watchdog status bit is set. Separate interrupt enable bits are available for flexibility.

Code of ADC and main functions implementation The main two functions are:

1. ADC_voidInit (ADC_CH, Sampling_time)

   Here we defined the channel which is connected to the analog voltage source, and the sampling rate:

   - ADC Sample Time
   - Select the alignment Converted data, (1 : Left Aligned)
   - select the external event used to trigger the start of conversion of a regular group (Software start: to trigger the ADC to start conversion)
   - Power On the ADC peripheral, (Setting ADON bit)

2. ADC_u16StartConversion (ADC_CH)

   Here we define the channel which is connected to the analog voltage source

   - Setting the channel number from ADC_SQRx register and the number of conversions (Here we use 1 conversion for the channel" Single Conversion")

- Start conversion of channel defined by software triggering
- waiting for the flag which is indicator for ending of conversion
- clearing the flag, to make it able to make another conversion from anther sample
- Reading the result "Converted data" from register of data (ADC_DR).

Fourth Element: We use a BUZZER to alarm the people at the building that there is Fire as it is sound when there is a voltage across it and its sound is proportional with the current passes through it.

It has Two terminals:

• Positive (+): which is connected to Vcc (3.3).

• Negative (-): which is connected to Ground.

Fifth Element: DC Motor:

Sixth Element: Wi-Fi Module which is defined in details in Section 3.6.

3.5.1.3 Algorithm

We can discuss the previous Flow chart as the following:

1. we define some values and name of pins which is known as Preprocessing section; and usually we do it as these values don't change through the run time and take no memory.

   Examples:

   #define ON_Fire_Value 650

   #define LED_PIN PIN5

2. we start the program and Configure the used pins as its output or input or analog input; using the functions of "GPIO" Peripheral.

   Example:

   GPIO_Set Pin Direction (GPIOB, LED_PIN, OUTPUT_PP_2MHZ)

Figure 3.17: Flow chart for Algorithm

3. we take the input from our sensors (MQ_2) and (Flame) and store it in variables to be processed by micro using the functions of "ADC" Peripheral.

   Example:

   Here we take the output of ADC channel which is connected with MQ_2 Sensor and put it in "MQ_2Value" variable.

   MQ_2Value = ADC_Start Conversion(ADC_Channel1)

4. We check if the valued converted by ADC ("MQ_2Value") is above the value of Firing and in case of:

   - Yes: We check again if there is flame from values of Flaming Sensors which are connected at analog pins with the micro controller and in case of:

     Yes: We have sure that there is Fire and hence we will take the following procedures:

     i. Send a message to the user to alert him.

     ii. Power on the led and buzzer to alert people.

     iii. Turn on DC Motor to open the window/balcony.

     iv. Turn off the Electricity using "relay"

     No: there is a lot of smoke gas but there is no fire hence we can only open the window:

     Turn on DC Motor to open the window/balcony.

   - NO: We check again if the valued converted by ADC ("MQ_2Value") is between the smoking value and in case of

     • Yes: Hence we will alarm the people at the building that the smoke gas increases:

     • Power on the led and buzzer to alert people.

     • No: There is no danger and

     • Power off the led and buzzer to alert people.

3.5.1.4 Circuit Diagram

We use Proteus program to show the circuit diagram of the Fire System at "ROOM1".

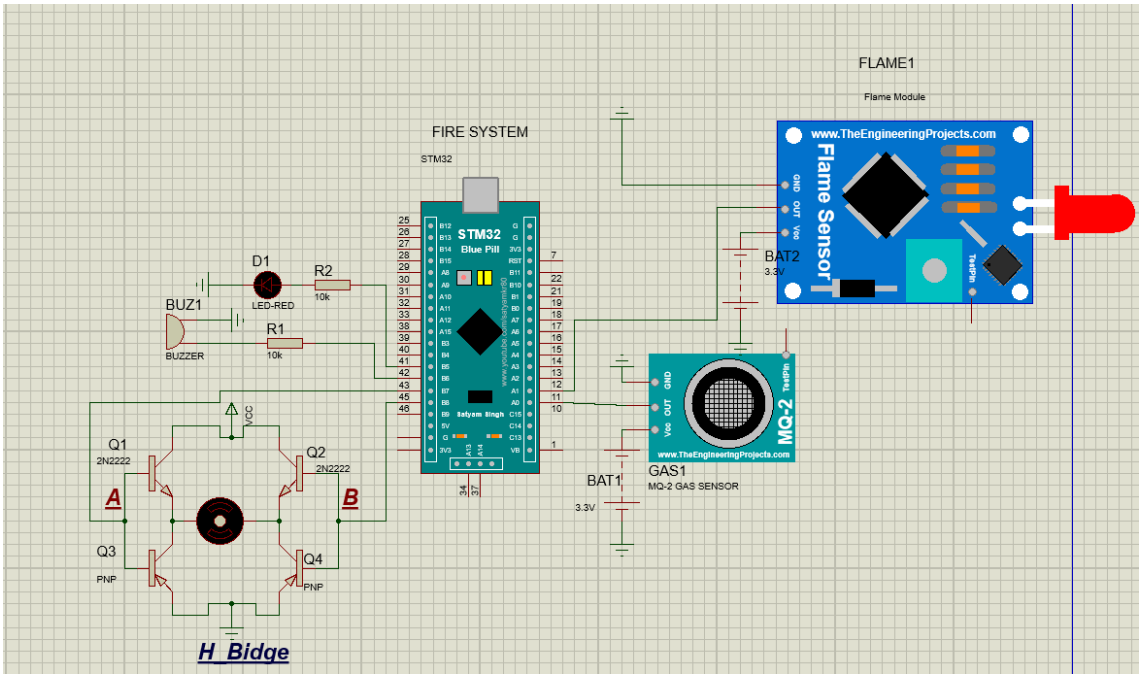

Figure 3.18: Circuit Diagram of Fire Subsystem

Note: we handle in our system two rooms but for simplicity we drew one room as the second room is a copy from this shown above.

Now we can discuss our circuit in details:

As the output of both the Flame Sensor and MQ_2 Sensor are analog we interface them with analog pins of micro (A0, and A1), and configure them as INPUT_ANALOG through the configuration Section in code.

Example: GPIO_ Set Pin Direction (GPIOA, Pin1, INPUT_ANALOG);

In reality we used Flame Sensor Not Flame Module but there is no library for flame sensor in "proteus"; but their results are same.

Real Connection for Flame Sensor:

• As the LED and BUZZER have low resistance we connect in series with both of them a resistance (300 ohm).

• Connection of DC_MOTOR:

As the DC Motor has high current and to control it with micro without ganger

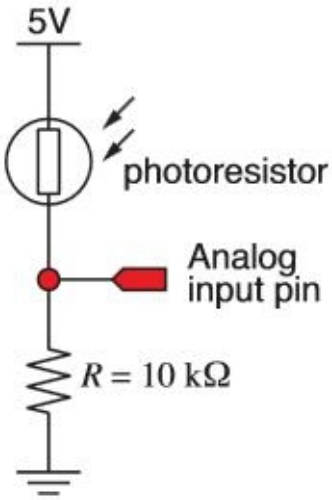

Figure 3.19: Connection of Flame Sensor

to it, we use the micro as Control Circuit not as a Power Circuit thus the micro isn't connected directly to the motor but we use a transistor as an electrical switch to the micro.

This transistor is a (2N2222) PNP transistor, and from data sheet of it we concluded that (V_BE minimum = 0.6 V) to work in saturation mode, and our micro outs 3.3V hence we inserted a resistor (R3, R4) between the output pin of micro and the base of transistor, But What is the Value of This resistor?!

- R = dV/I , where: dV=3.3- 0.6=2.7 V
- And the source Current of Micro is I = 25mA

Hence R = 2.7/25mA=108 ohm

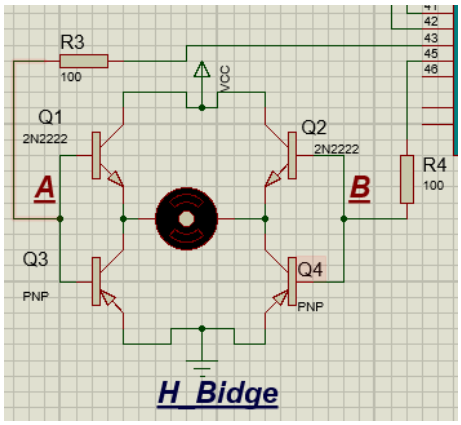

Figure 3.20: H- Bridge of DC – Motor

Table 3.2: status of DC Motor

PIN B6 (A)	PIN B7 (B)	Motor Rotation
LOW	LOW	Grounded from two terminals "Stop"
LOW	HIGH	Counter Clock Wise
HIGH	LOW	Clock Wise
HIGH	HIGH	Powered from two terminals "Stop"

Thus the following table shows the operation of H_ Bridge and how can it work with only two pins from micro These two pins woks as output_pp.

Pin A: connected with B6 of micro.

Pin B: connected with B7 of micro.

Hence we can stop Motor and move it in the both direction using only two pins with micro.

3.6 Third Subsystem (Harmful Gases)

3.6.1 Introduction

Here our aim is to protect the user from harmful gases which may leak from sources of natural gas around him or and instrument worked by natural gas like oven or stove, and from CO gas which leaked from heaters which work by natural gas; hence we have two main sensors which are MQ7 and Mq4 which sense CO and natural gases, and we use STM32 as a micro controller to process the values of these sensors and then make suitable actions like turning off sources of gas and the IOT technology to send messages to alert user when there is leakage of gases, or to take orders from user to turn off the main source of Natural gas of the building.

3.6.2 Block Diagram of Harm Gases Subsystem

First Element: Source of CO gas or Natural gas.

As Natural gas may be leaked from stoves/ovens in kitchen.

And CO gas may be leaked from heaters in bath room.

Figure 3.21: Block Diagram of Harm Gases

Second Element:

- MQ – 4 Natural gas sensor

  Sensitive material of MQ-4 gas sensor is SnO2, which with lower conductivity in clean air. When the target combustible gas exists; the sensor's conductivity is higher along with the gas concentration rising. Please use simple electro circuit, Convert change of conductivity to correspond output signal of gas concentration.

  MQ-4 gas sensor has high sensitivity to Methane, also to Propane and Butane. The sensor could be used to detect different combustible gas, especially Methane, it is with low cost and suitable for different application.

  It also has many features which made it suitable for our System:

  Good sensitivity to Combustible gas in wide range.

  High sensitivity to Natural gas from 300 ppm to 10000 ppm.

Operating Voltage is VCC = ((3:5) V)" DC"

Long life and low cost

Analog output voltage: (0V to VCC V) "Which is used for our system" according to the concentration of gas.

- MQ – 7 CO gas Sensor

  Sensitive material of MQ-7 gas sensor is SnO2, which with lower conductivity in clean air. It makes detection by method of cycle high and low temperature, and detect CO when low temperature (heated by 1.5V). The sensor's conductivity is higher along with the gas concentration rising. When high temperature (heated by 5.0V)

  It also has many features which made it suitable for our System:

  - Good sensitivity to Combustible gas in wide range
  - Detection of concentration: 10-1000ppm CO
  - Heater voltage: 5V±0.2V AC or DC(high), 1.5V±0.1V AC or DC(low)
  - High sensitivity to CO gas

Third Element: STM32F103C8T6 Micro-controller. As we mentioned before we will use ARM Cortex M3 (STM32F103C8T6) Micro-controller to Process the data collected from sensors and the other components and then take the correct actions corresponding to the algorithm of every subsystem.

Peripherals Needed for This Sub System From STM32 STM32 CortexM3 contains All the needed peripherals which will be used to handle the Systems; and the peripherals which are used through this system are:

- RCC. "Reset Control Clock"
- GPIO, "General Purpose Input Output"
- ADC, "Analog to Digital Converter"
- SysTick
- Timer (Tim4)

• USART1, For IOT.

• NVIC, Nested Vectored Interrupt Control.

And all of them are discussed in details in the previous sections.

Fourth Element:

• Servo Motor, which is used to turn off the source of natural gas of Heater/Stove, and it is discussed in the Security System.

• Buzzer: is used to alert people in the building by its sound.

• Wi-Fi Module which is defined in details in 3.5 section

Fifth Element: Servo motor which is connected with the General Key of Natural gas in the building and it can be controlled only by the user to turn off this key by sending a message through Wi-Fi to the micro

3.6.3 Algorithm of Harm Gases Subsystem

1. First

    - Pin Configurations

      a) we define some values and name of pins which is known as Preprocessing section; and usually we do it, as these values don't change through the run time and take no memory.

      Examples:

      #define MQ4_Threshold_Value_mv 1400

      #define MQ7_Threshold_Value_mv 1650

      Hence we can express MQ4_Threshold_Value_mv as 1400 to make some calculations according to this value.

      b) Defining The direction for every pin if it is input or output using this function.

      GPIO_Set Pin Direction (GPIOB, LED_PIN, OUTPUT_PP_2MHZ);

    - Clock Initializations

      We enable the clock for the processor using this function

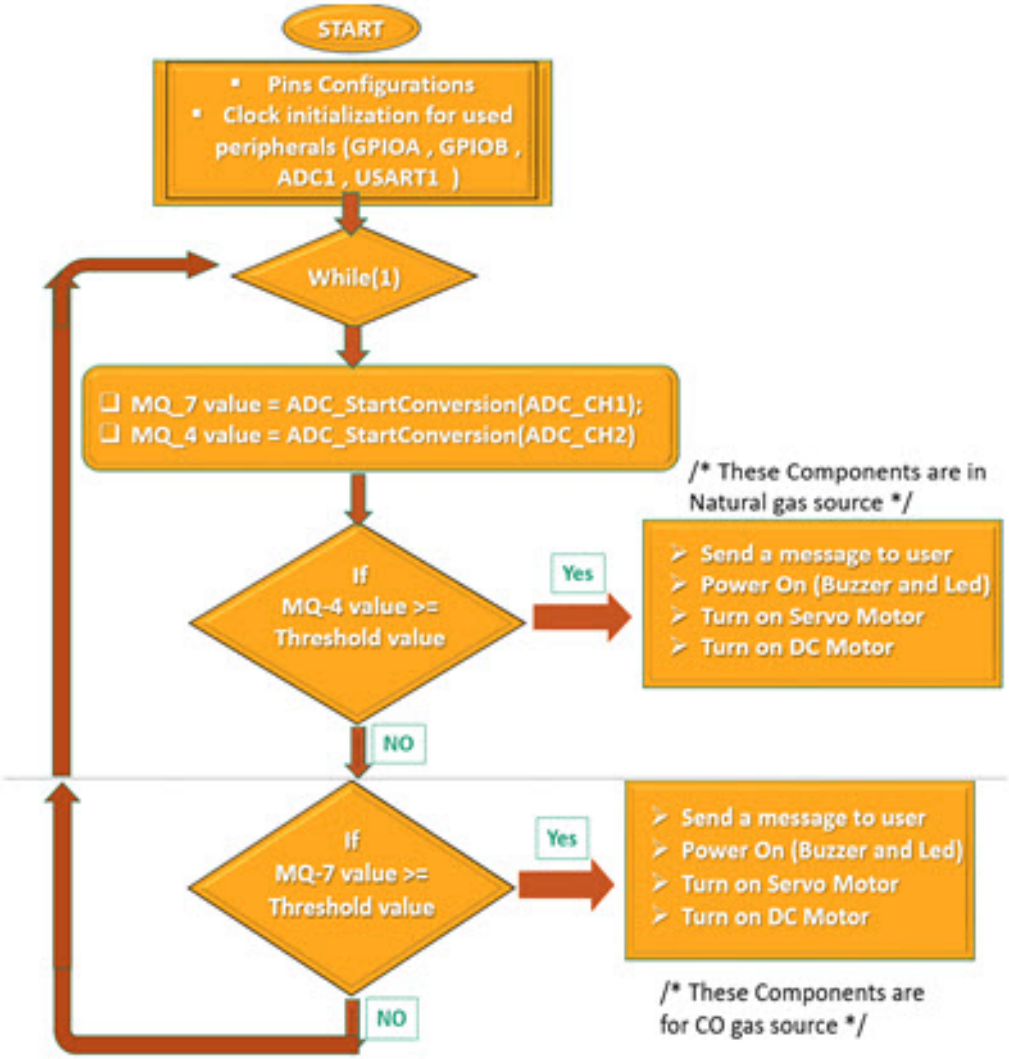

Figure 3.22: Flow Chart of Harmful Gases

RCC_void Clock Sys Init ();

And Enable the clock for each peripheral we will use on its bus using this function:

RCC_void Enable Clock (RCC_APB2, GPIOA);

2. we take the input from our sensor (MQ_4) and store it in variable to be processed by micro using the functions of "ADC" Peripheral.

   Example:

   Here we take the output of ADC channel which is connected with MQ_44 Sensor and put it in "MQ_2Value" variable.

   MQ_4Value = ADC _ Start Conversion (ADC_Channel1);

3. Always We will check if the valued converted by ADC ("MQ_4Value") is above the value of Threshold and in case of:

   • Yes: We will execute the following actions:

   • Send a message to user to alert him that there is leakage of natural gas.

   • Power On (Buzzer and Led)

   • Turn on Servo Motor to turn off source of gas

   • Turn on DC Motor to open Window

   No: we check for presence the CO gas using MQ7 Sensor and in case of:

   • Yes: We will execute the following actions:

   ◦ Send a message to user that there is leakage of CO gas.

   ◦ Power On (Buzzer and Led)

   ◦ Turn on Servo Motor to turn off source of gas

   ◦ Turn on DC Motor to open Window

   • No: go to the start of While loop to check again for the presence of natural gas.

3.6.4 Circuit Diagram of Harm Gases Subsystem

We use Proteus program to show the circuit diagram of the Fire System at Bath room (for Heater) and Kitchen (For Oven/Stove).

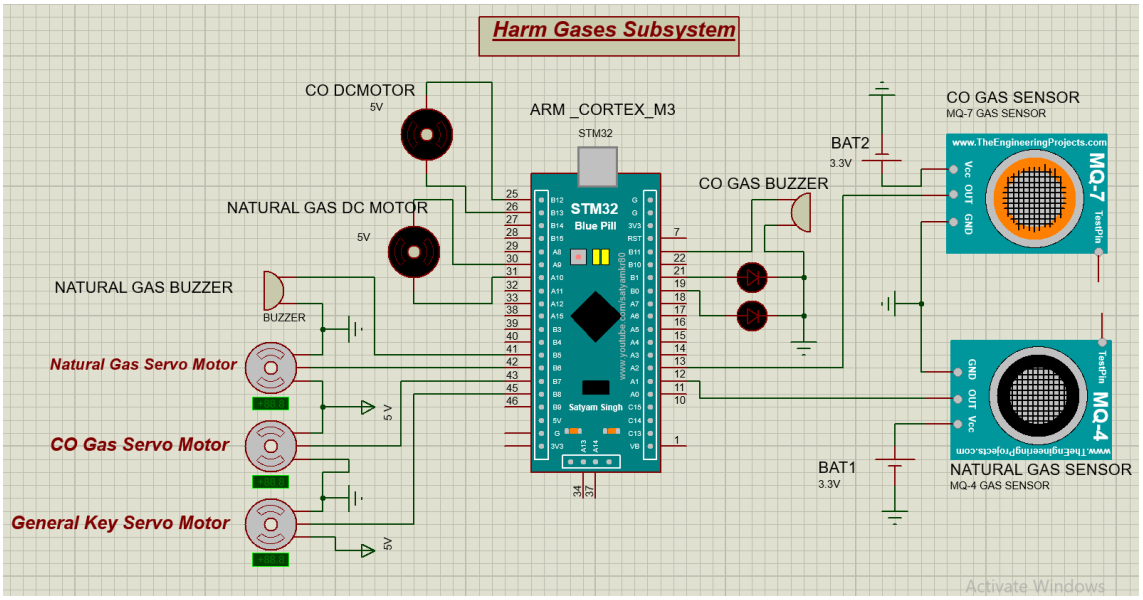

Figure 3.23: Circuit Diagram for Harm Gases

Notes:

There are three Servo motors as we have three Keys for controlling The Source of natural gas:

- Key in bath room.
- Key in Kitchen
- General key.

3.7 IoT Module

3.7.1 Introduction

Smart Homes, also known as automated homes, intelligent buildings, integrated home systems, are a recent design development. Smart homes incorporate common devices that control features of the home. Originally, smart home technology was used to control environmental systems such as lighting and heating, but recently

the use of smart technology has developed so that almost any electrical component within the house can be included in the system. Moreover, smart home technology does not simply turn devices on and off, it can monitor the internal environment and the activities that are being undertaken whilst the house is occupied. The result of these modifications to the technology is that a smart home can now monitor the activities of the occupant of a home, independently operate devices in set predefined patterns or independently, as the user requires [3, 11].

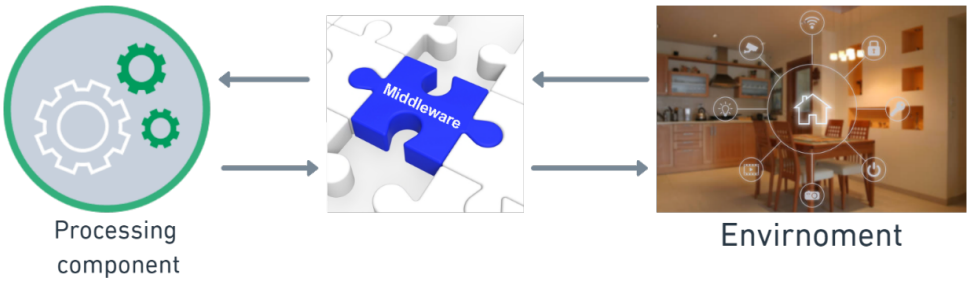

Figure 3.24: Smart Home

As shown in Figure 3.24, the IoT module includes three main components. The first component is "Environment" which corresponds to the real-world home. The second component is "Middleware" which responsible for managing and controlling the attached devices in the environment. The last component is "Processing" which responsible for manipulating the data send and received from the environment.

3.7.2 IoT Module Block Diagram

Figure 3.25, Represents the IoT module block diagram which consists of different parts.

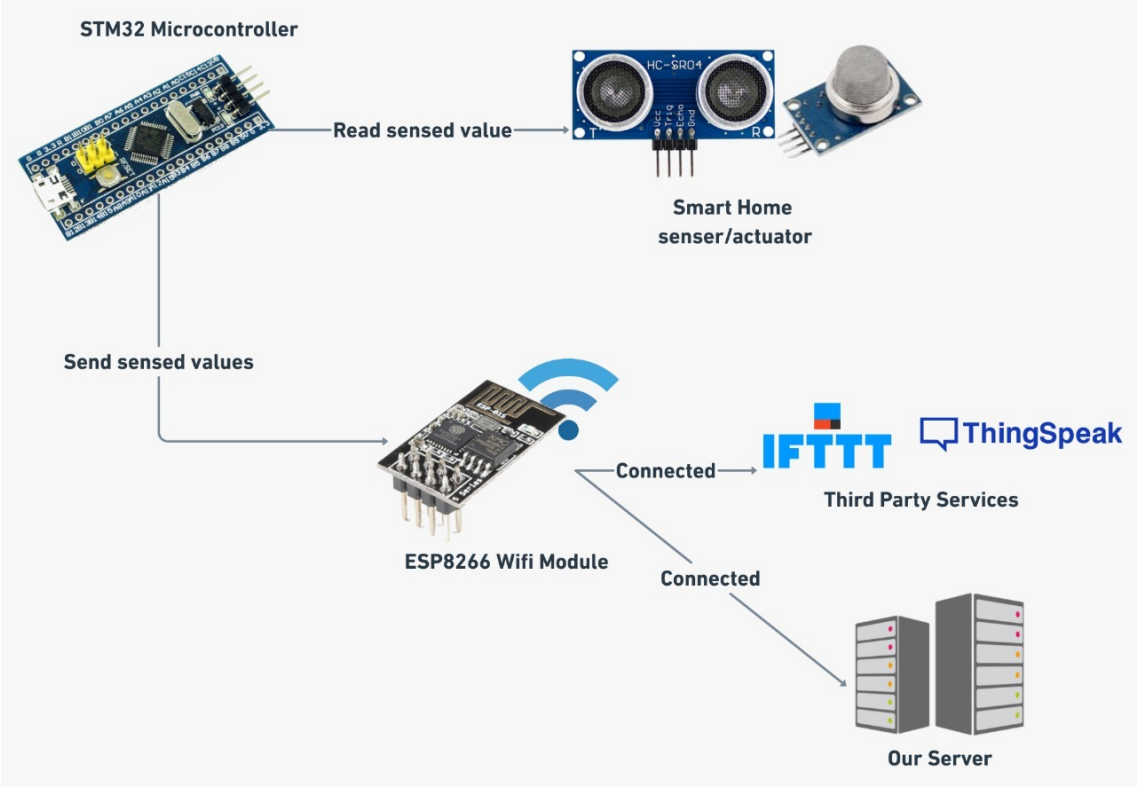

Figure 3.25: Block Diagram

By STM32 peripherals (RCC, GPIO, SYSTICK, NVIC) microcontroller receive sensors reads in his GPIO pins, then STM (GPIOA9:TX, GPIOA10:RX) send data to Esp8266 by Usart Protocol.

"The ESP8266 is the name of a microcontroller designed by Espressif Systems in china. The module can even be programmed to act as a standalone WiFi or it can be used as a WiFi module suitable for adding WiFi functionality to an existing microcontroller project via a UART serial connection.

The ESP8266 WiFi module and the microcontroller can be interface through USART and with the help of a wide range of AT Commands, the Microcontroller can control the ESP Module, ESP8266 is act as a microcontroller so it has flash memory on it , this flash memory contain of firmware(any software that manage hardware), this firmware understand group of AT Commands (some commands that start with AT), The AT Commands of the ESP8266 WiFi Module are responsible for controlling all the operations of the module like restart, connect to WiFi, change mode of operation and so forth.

Table 3.3: AT Commands

Command	Description	Response
AT	Test AT startup	OK
AT+CWMODE=1 AT+CWMODE=2	station mode AP mode	OK
AT+CIPMUX=0 AT+CIPMUX=1	single connection multiple connection	OK OK
AT+CWJAP=”SSID”, “Pass”	Connect to AP	Connected or Not
AT+CIPSTART=”Mode”, ”IP”, Port number	Connect to Server	Ok or Error
AT+CIPSEND=”Length”	Send length of Data	Ok or Error

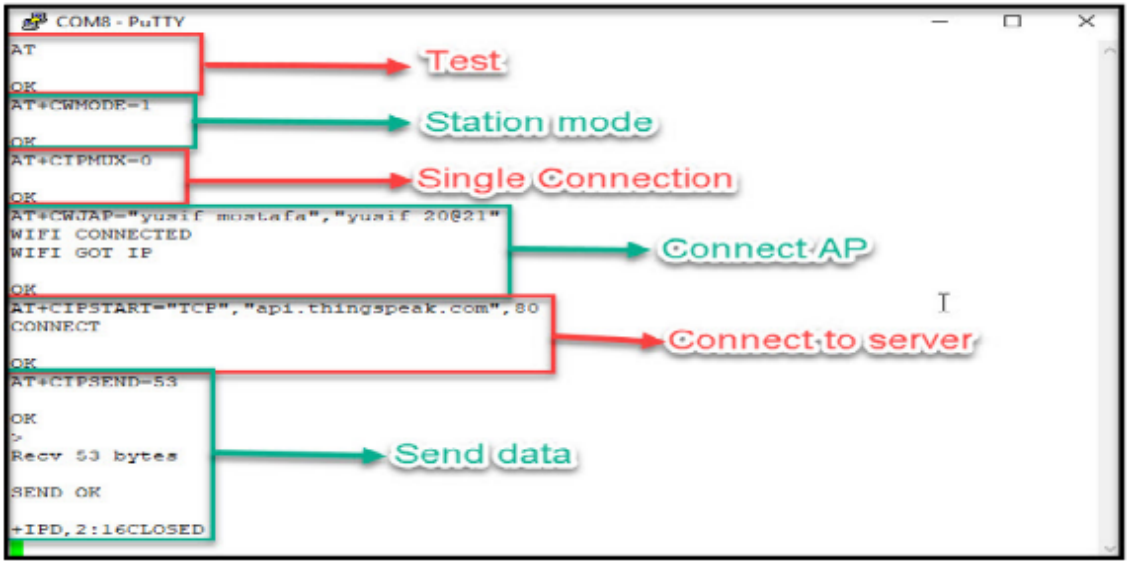

Figure 3.26: AT Commands

- Features of ESP8266:

  (1) Integrated low power 32-bit MCU

  (2) Integrated 10-bit ADC

  (3) Integrated TCP/IP protocol

  (4) Support USART Protocol

(5) Baud rate 115200 (speed of bit/sec)

- Connection of ESP8266(as a WIFI module) with STM32

Table 3.4: Connection of ESP8266

ESP8266	STM32	VCC / GND
TX	RX(GPIOA10)	
CH_EN		3.3V
RST(active law)		3.3V
VCC		3.3V
GND		GND
RX	TX(GPIOA9)	

USART The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange, The USART offers a very wide range of baud rates using a fractional baud rate generator. It supports synchronous one-way communication and half-duplex single wire communication. It allows multiprocessor communication .

- Specifications of USART:

  - Wire Connection

  - Serial Connection

  - Peer to peer

  - Asynchronous

  - Data direction (Full duplex)

  - 115200 baud rate (speed) need to connect with WIFI module

- Frame format of USART:

  - Idle state (high)

  - Start bit (1 bit_low)

- Data bits (8 : 9 bits)

- Parity bit (1 bit for check data)

- Stop bit (1: 2 bits_high)

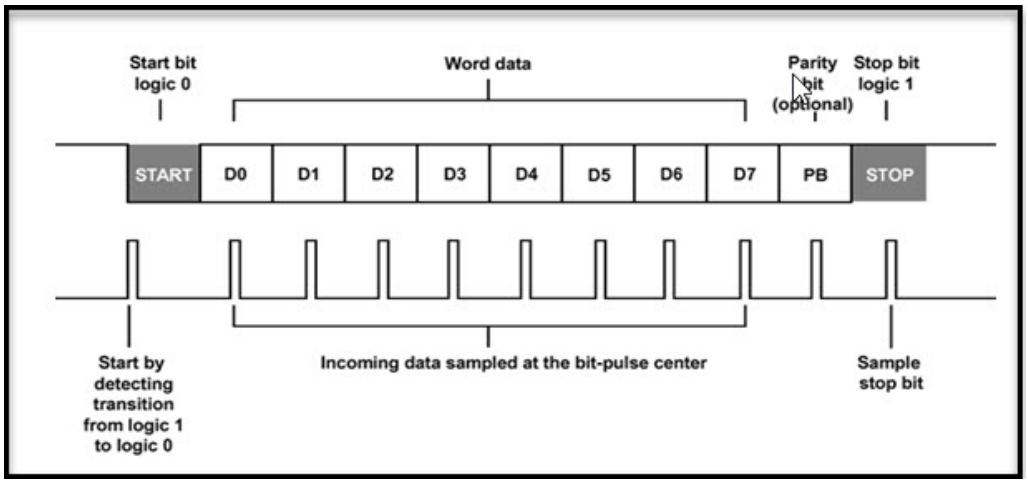

Figure 3.27: Frame of UART

3.7.3 Algorithm of data transmission/receive by USART

Data transmitted/received is stored in a 32-byte FIFOs, so to transmit or receive data we should follow some steps and shown in Figure 5.

(1) Initialize Baud rat

(2) Enable Status register, SR

(3) If word length = 8 bit

Disable word length bit in control register

Else If word length = 9 bit

Enable word length bit in control register

(4) Enable/Disable control register parity and interrupt bits

(5) Enable/Disable control register stop bits

(6) Send data on data register

(7) Read data from data register

WIFI Module connect to ThingSpeak platform and send data of sensors on it, when sensors sense any action for example high gas rate), ThingSpeak take reaction and send SMS android message to user by connect with URL of IFTTT platform.

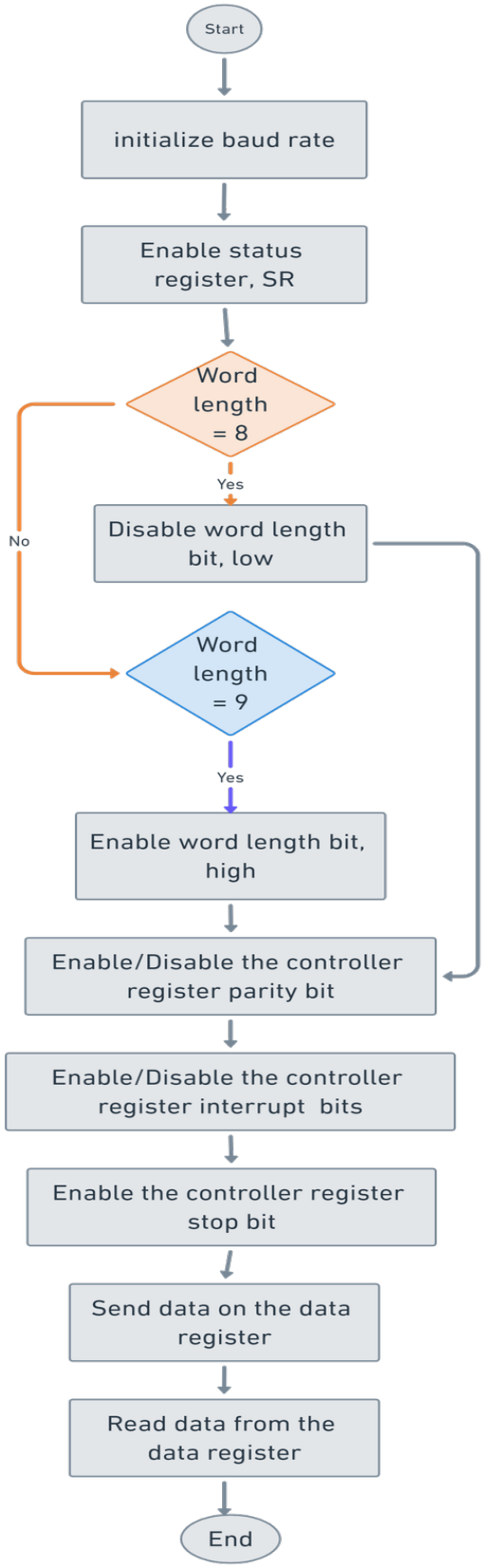

Figure 3.28: Flaw chart of UART Algorithm

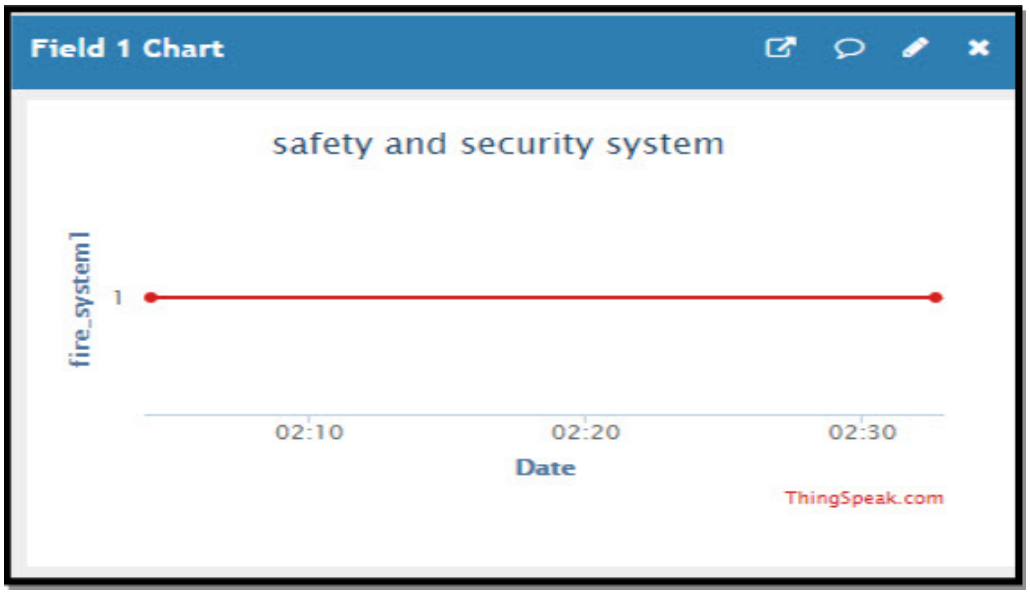

Figure 3.29: Sensor read in ThingSpeak

Figure 3.30: SMS send to user

ThingSpeak is an IoT analytics platform service that allows to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by devices to ThingSpeak.

IFTTT is short for "If This Then That" and is the free way to do more with hundreds of the apps and devices , We bring services together into Applets and connections. These are specific things that can happen when you connect services.

After user receive SMS message , go to his server this server connect to stm32 microcontroller to control everything in his home and know the degree of temperature , humidity and gas on the home.

- Turn on/off gas source
- Turn on /off doors and windows

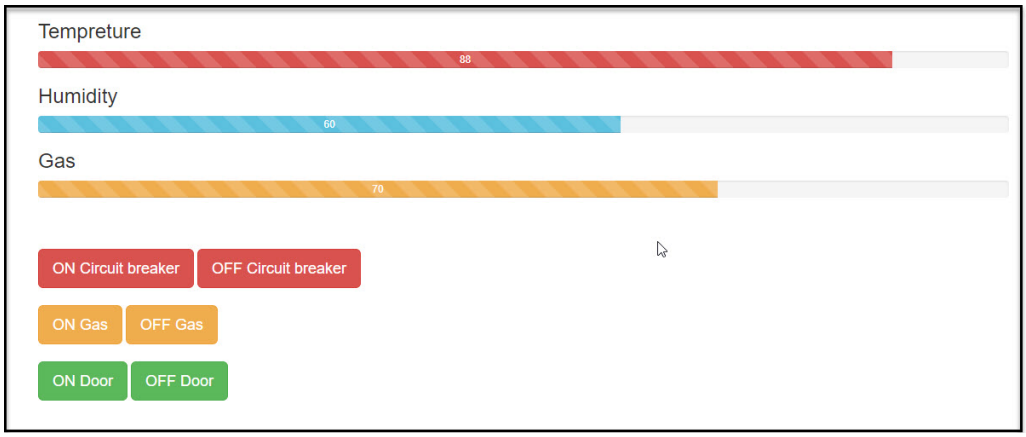

Figure 3.31: User Server

- Turn on/off electric source (circuit breaker)
- Turn on buzzer to warn his family

3.7.4 Circuit diagram of IoT module

Figure 3.32, represents the circuit diagram for the IoT module. In the following paragraphs, we will describe its components.

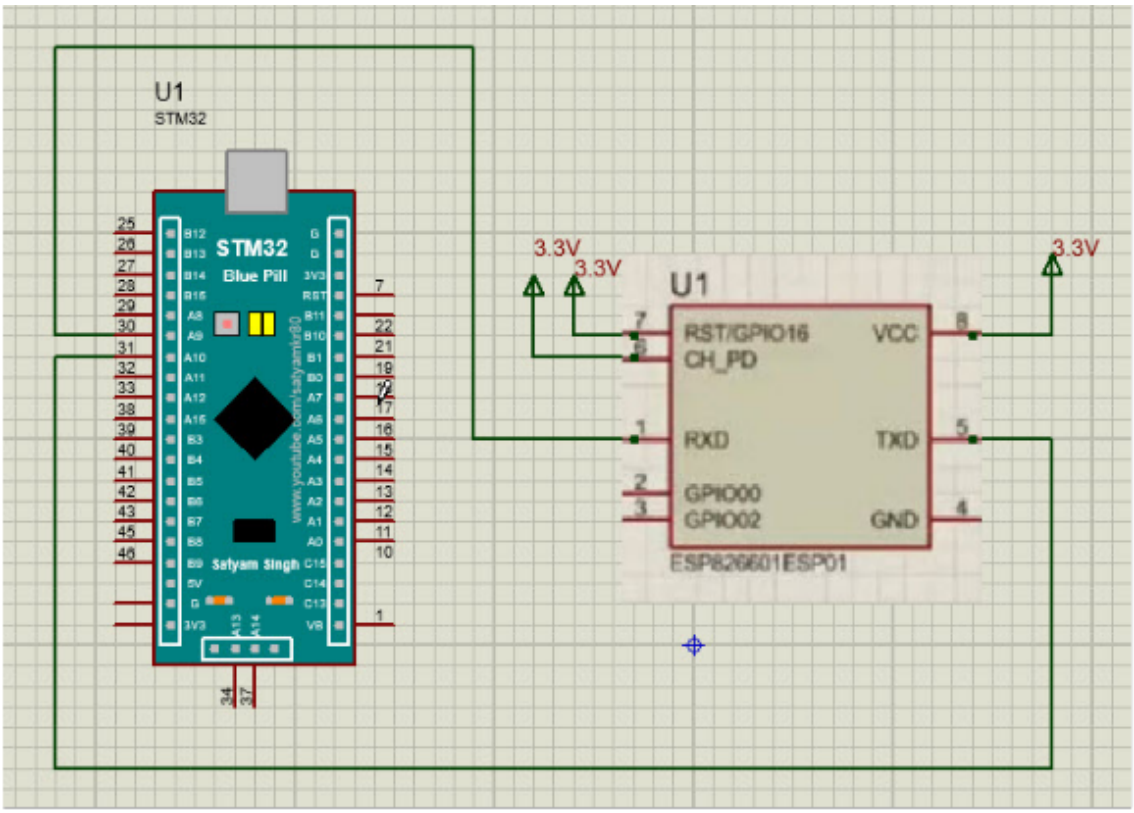

Figure 3.32: Circuit Diagram of IoT Module

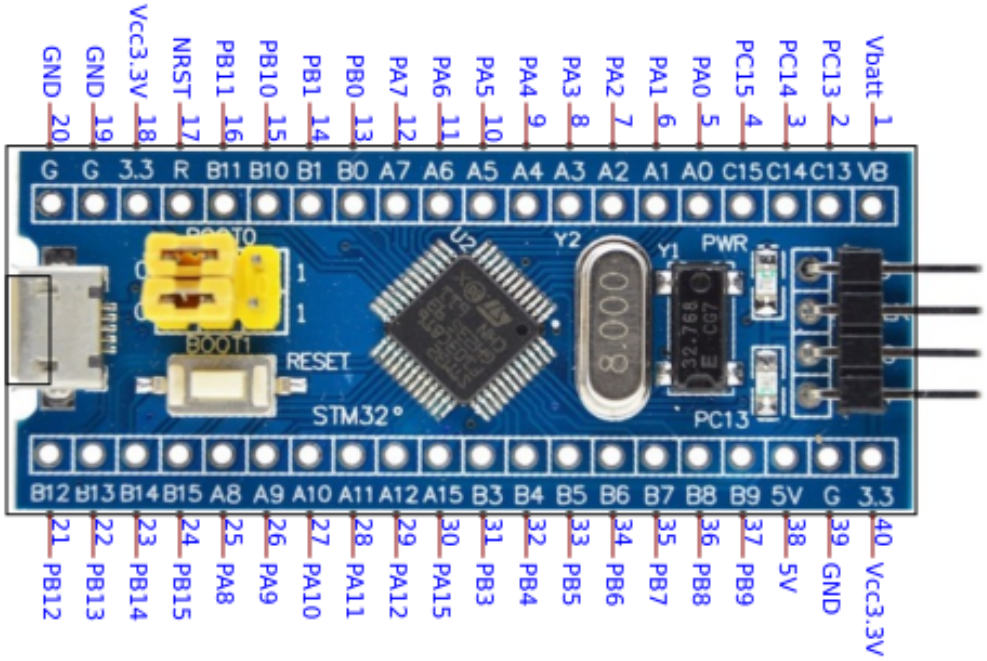

Figure 3.33: STM32 Microcontroller

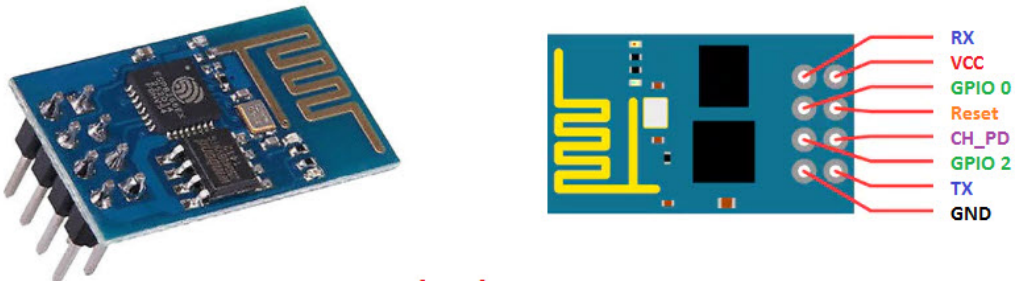

Figure 3.34: ESP8266 (WIFI Module)

STM32 Microcontroller

STM32 is the microcontroller processor that control all the circuit, read sensors, send and receive data from ESP8266 [12].

ESP8266

ESP8266 is a wifi module act as wifi connect with access point in the home, receiving data from STM32 and sending it to middle area (platforms and servers) .

3.7.5 Algorithm of iot module

As shown in Figure 3.35, to design the circuit program we need to follow some steps.

1. Initialize ESP
2. Connect to wifi using SSID and Password
3. Connect to server with TCP
4. Send TCP data length
5. Check and send natural gas value
6. Check and send carbon oxide gas value
7. Check and send ultrasonic sensor value (stealing system)
8. Check and send fire in room1 value
9. Check and send fire in room2 value
10. Receive SMS message
11. Send user reaction to txt file
12. Send current of temperature, humidity and gas values

3.8 Boot-loader Design for Microcontrollers in Embedded Systems

3.8.1 INTRODUCITON

Microcontrollers have proliferated into every nook and cranny of our daily lives from simple 8-bit devices that control our toaster ovens to powerful 32-bit DSP's that provide us with the rich media and entertainment that we have all become accustomed to. Without microcontrollers, our lives would not only be less exciting but we would lose a level of control over our world that we can. no longer live without. Billions of microcontrollers are sold each year with the number continually climbing [2].

What happens to these microcontroller based products when millions of units have shipped and a software "enhancement" needs to be made? Does every unit need to be returned to the manufacturer every time the software is updated? Are televisions, blue-ray players and other devices returned to the manufacturer peri-

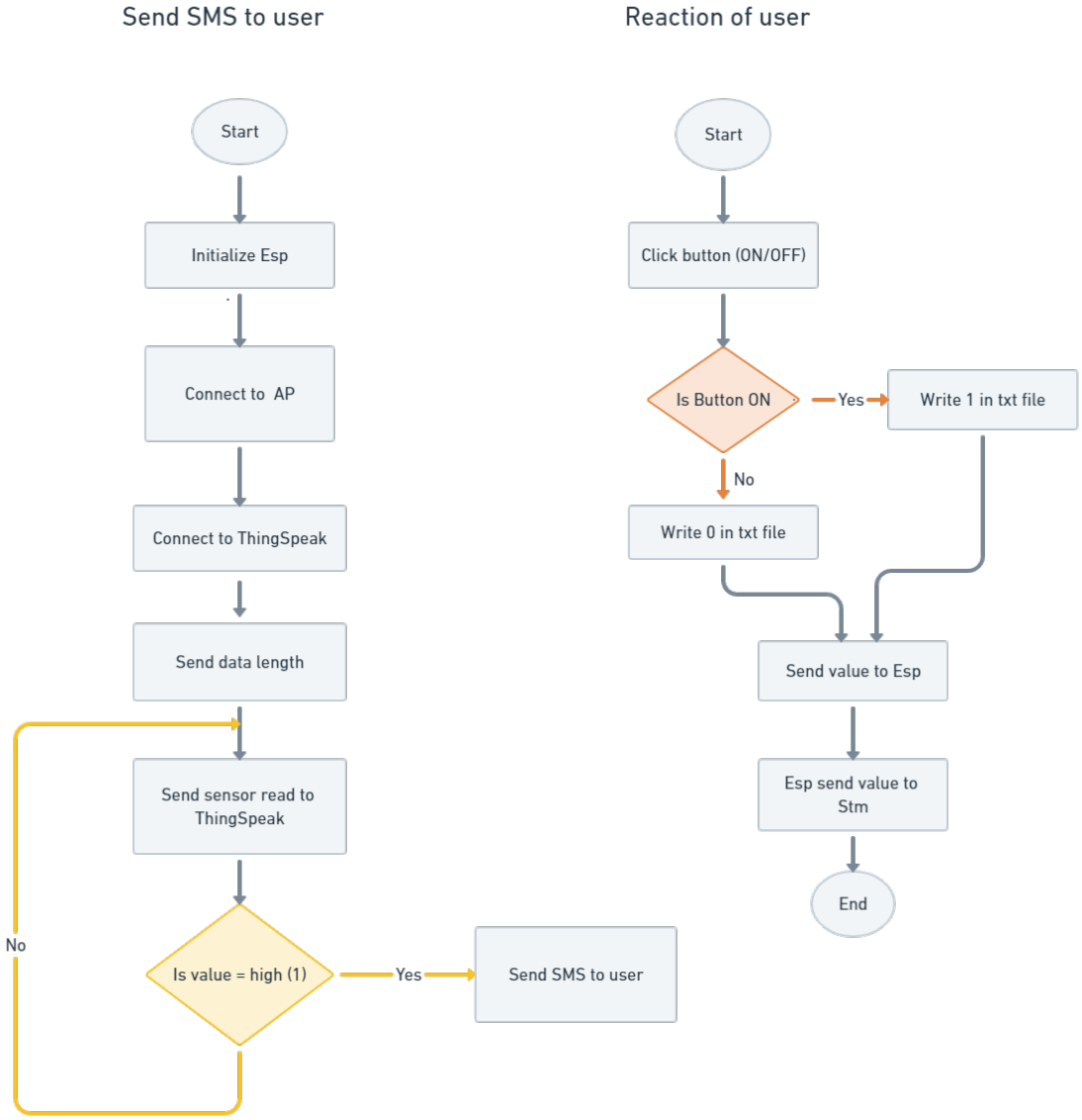

Figure 3.35: Flaw Charts send SMS to use & take react

odically so that the customer can continue to have the latest and greatest software operating on their device? The obvious answer to these questions is absolutely not and the primary reason why is that most systems ship with a boot-loader on-board [8].

A boot-loader is an application whose primary purpose is to allow a systems software to be updated without the use of specialized hardware such as a JTAG programmer. In certain cases, it can also be the earliest point at which the integrity of an embedded system can be checked. The boot-loader manages the systems images. There are many different sizes and flavors to embedded boot-loaders. They

can communicate over a variety of protocols such as USART, CAN, I2C, Ethernet, USB and the list goes on for as many protocols that exist. Systems with boot-loaders have at least two program images coexisting on the same micro-controller and must include branch code that performs a check to see if an attempt to update software is in progress.

During the initial stages of product development it is very common for the boot-loader to be overlooked by the development team. The primary reason for this is that the boot-loader is not the primary end product that is going to be sold to the customer but the boot-loader is potentially the most important part of that product. The boot-loader allows a company to launch their product with software that only fulfills a portion of their final feature set and then add features to their product once it has been launched into the market. It also allows them to fix bugs that are discovered after the system has been released into the wild.

For an embedded software engineer, a boot-loader requires a full understanding of how a processor works, how to utilize its memory and how to work on the processor at the lowest levels. Boot-loader development can be an extremely challenging endeavor to undertake but absolutely a rewarding one. Once a developer has gone through the process, each additional boot-loader becomes easier and easier to implement by following a common and consistent approach to boot-loader design.

3.8.2 BOOT-LOADER REQUIREMENTS

Before any software project is undertaken, it is always recommended that at a minimum some informal process of requirements gathering be performed. Requirements dictate to the developer what it is that they are going to be designing. Requirements can serve as a roadmap in the design process. The IEEE Computer Societies Software Body of Knowledge (SWEBOK) defines a software requirement as "a property which must be exhibited in order to solve a specific problem" (Society, 2004). The requirements tell the software developer key properties that the software must exhibit in order to present an appropriate solution.

There is no such thing as a complete list of project requirements from a customer. Project requirements tend to be fluid and often change throughout the design process. Modern day software processes such as Agile take these changes into effect and break the design process up into iterations of a fixed period of time. At the start of each iteration, the requirements are reviewed with the customer, allowing updates and changes to be made in addition to highlighting the critical path for the iteration. The key components of successful requirements gathering and execution include the following.

- Requirements Elicitation
- Requirements Analysis
- Requirements Specification
- Requirements Validation and Traceability

Each boot-loader will have its own unique set of requirements based on the type of application; however, there are still a few general requirements that are common to all boot- loaders. These requirements can be broken down into seven fundamental groups of requirements that are common to the boot-loader. They are

- Ability to switch or select the operating mode (Application or boot-loader)
- Communication interface requirements (USB, CAN, I2C, USART, etc)
- Record parsing requirement (S-Record, hex, intel , toeff , etc)
- Flash system requirements (erase, write, read, location)
- EEPROM requirements (partition, erase, read, write)
- Application checksum (verifying the app is not corrupt)
- Code Security (Protecting the boot-loader and the application)

3.8.3 THE BOOT-LOADER SYSTEM

Boot-loaders can come in many different sizes and in many different flavors but in general the operation of a system with a boot-loader is relatively standard. There are three major components to these systems that can be seen in Figure 3.36. They are the branching code (green), the application code (blue) and the boot-loader code

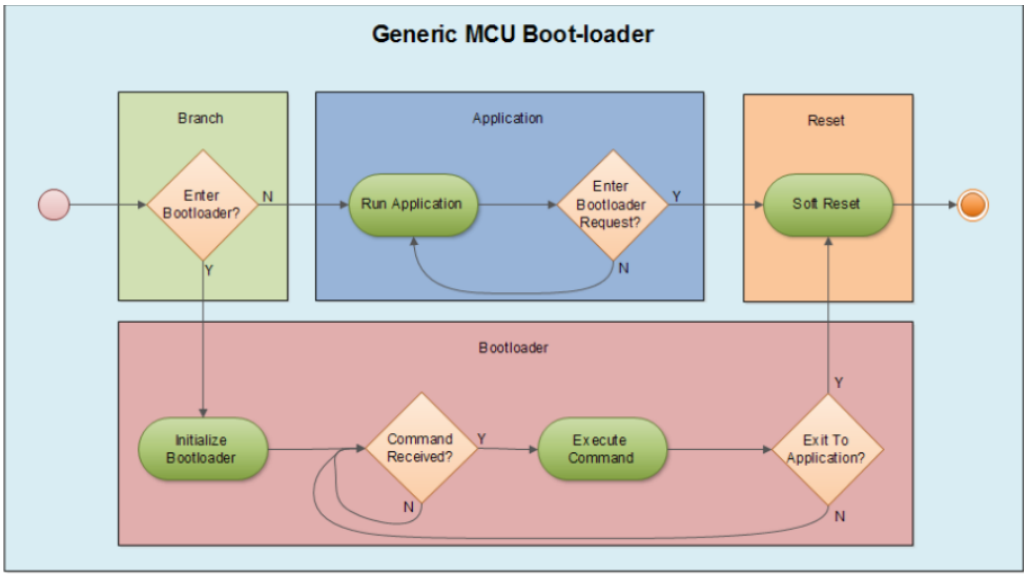

Figure 3.36: General boot loader

(red). For most systems we prefer them to be executing the application path the majority of their operational life. Figure 1 highlights the execution path to get to the application by the dashed red line. The orange block is a common block used by both the boot-loader and the application to reset the system.

The branching code (green) makes the decision as to whether the boot-loader is loaded or whether the application is loaded. The branching code in a very simple system can be nothing more than checking a GPIO for a certain state to enter the boot-loader; however, such basic systems are usually only used by chip manufacturers in order to demonstrate the functionality of the chip and is usually not recommended if a robust boot-loading solution is required. In a more complex system, the boot-loader may completely load itself into memory and perform some basic system functions for a short period of time before making the branching decision possibly to perform a system integrity check. For this reason, in most cases the branching code is included in the boot-loader.

The application code (blue) is executed only after the branching code has made the decision that there are no requests to enter the boot-loader and that the application code is safe to execute. Even while the application is loading and performing its duties, it should be designed so that it can accept a command to enter the boot-loader. When the application receives such a request the application will perform

any cleanup that needs to be performed and when it is safe to do so the application will perform a soft reset of the system. In most cases this is done by writing incorrect values to the watchdog timer that in turn will then cause a power-on-reset of the system.

The boot-loader code (red) performs all of the functions of the boot-loader. When the boot-loader is loaded into memory it will begin initializing a basic subset of peripherals that are required in order for the boot-loader to perform all of its functions. These peripherals are typically the system clock, interrupt service routines and tables, a communication peripheral and perhaps a state-machine or basic task scheduler. This allows the boot-loader to communicate with the outside world and accept commands to perform flashing instructions.

3.8.4 BOOT-LOADER BEHAVIOR

A boot-loader in itself is not that different from a standard application; in fact, it is a standard application. What makes a boot-loader special is that it is sharing flash space with another application and has the capability to erase and program a new application in its place. Like every other application, one of the first priorities of the boot-loader is to initialize the processor and the minimum number of peripherals required to carry out boot-loading functions. It's best to keep the use of peripherals to as few as possible in the boot-loader in order to attempt to maximize the flash space that will be available for the application code.

There are two behavioral models that describe how a boot-loader can behave. In the first model, the boot-loading process is completely automated and self-contained within the system. An example of this would be an SD card boot-loader. The boot-loader would automatically detect the new firmware and manage its own flashing process. Commands from an external source would not be required to successfully carry out the boot-loading process.

In the second model, the system does not automatically handle the boot-loading process itself. Instead, the boot-loader initializes into an idle state and awaits in-

structions from an outside source. This source would typically be a pc based software application that commands the boot- loader into the different states necessary to flash a new image onto the system. The primary reason for the external software application to command the process is that in most applications without an SD card, there is not sufficient space to retrieve the whole software image. Instead, an external source with the image acts as the master of the boot-loading process. This paper will focus on the later model.

Each boot-loader will be unique in the number and type of commands that are supported. A sophisticated boot-loader will have more commands than a simple boot-loader. Flash usage for the boot-loader increases with each command that is supported potentially resulting in less space for the application image. At a minimum the boot-loader should support three commands.

- Erase the flash – Removal of the application image from memory
- Write flash – Addition of a new application image to memory
- Exit / Restart – reboot with the intention of entering the application code

With these three commands all basic functions of writing an image to flash can be accomplished. There are more sophisticated commands that can be implemented that would greatly enhance the capabilities of the boot-loader. A few examples are

- Unlock the unit – enter a security key to allow flash to be erased and then written
- Erase EEPROM – erase configuration data stored on the system
- Write EEPROM – write configuration data to the system
- Read EEPROM - verify configuration data from the system
- Read flash – verify that the application image is correct
- Image Checksum – calculate an image checksum that can be checked for corrupted application
- Lock the unit – enter security mode that prevents flash from being written

Locking and unlocking the system provides added robustness and security to the system. It prevents reading the application image out of the system in addition to

accidental erasing or writing of the flash. EEPROM functions can be useful in the boot-loader particularly during the end- of-line of the manufacturing process. The boot-loader can act as a special mode in which to program the systems configuration data during the initial start-up of the system. The image checksum adds robustness to the system by allowing the system to verify that nothing has happened to the application image and that it is still completely intact.

The typical sequence for programming a device can be found below:

- Open image flashing tool (if supported)
- Start the boot-loader
- Erase the flash
- Send binary file information to the boot-loader (to be discussed later)
- Generate Checksum
- Quit the boot-loader and enter the application

3.8.5 BEHAVIOR

The behavior of the application image is of for the most part not of any interest to the boot- loader designer except in one aspect; the application needs to be capable of receiving a command to enter the boot-loader. This means that the application needs to have two boot-loader like capabilities :

- Set a piece of information that the boot-loader can detect to enter boot-load mode
- Reset the system to initiate a branching decision

The best place for the application to store a value that can be detected by the branching code is an EEPROM configuration value. EEPROM in most cases exists in a memory space that can be shared by both the boot-loader and the application. When the application receives a request to enter the boot-loader, the application can write a value to EEPROM and then perform the second function which is to reset the system.

The reset of the system is performed by entering an infinite loop, allowing the

watchdog timer to reset the system or by purposefully writing incorrect values to the watchdog register to force a software reset. The preferred method is to write incorrect values to the watchdog register because it guarantees that the system will be restarted immediately and minimizes delays.

3.8.6 START-UP BRANCHING

When the system starts up, there are at least two different software images that can be loaded and executed by the micro-controller, the boot-loader, the application and possibly a back- up application image. It is therefore necessary that as part of the boot-loader image code a branching algorithm be included that can handle the decision making process of which image to load.

There are many different methods that can be used to decide which image to load. The simplest method that can be used and is most often used in example code from chip manufacturers is the use of a GPIO line to make the decision. For example, if the GPIO signal is high, load the application; if it is low then load the boot-loader. This method is very simple and is often implemented using assembly language in order to quickly make the decision and jump to the image. Listing 1 shows a simple example for the Free scale S12X. In this example, if the logic signal checked is grounded, the top of the stack is loaded and program flow jumps to the boot-loader main function. If the logic signal is high, the application reset vector is loaded and then executed.

Closer examination of Listing 1 will reveal that there is a potential problem. What happens if the boot-loader has been loaded onto the micro-controller but an application has not yet been loaded and there is a request to enter the application? The code in Listing 1 will jump to the application anyways and jump right off into the weeds. This leads to one of the most important code branching checks; verification that the application reset vector exists.

3.8.7 MEMORY PARTITIONING

Every microcontroller has some form of non-volatile memory that is used to store the program. The most commonly used type of memory is flash. Flash is broken up into divisible sections. The smallest section of flash is often referred to as a page. Pages are organized into larger structures known as sectors. Sectors are in turn organized into larger structures known as blocks. Each microprocessor is different as to how these sections of flash can be manipulated. Most will allow you to write a single byte to flash at a time. Others may require that 8 bytes or 256 bytes be written at one time. In most cases, the smallest section of flash that can be erased at a time is a single sector that often consists of 4kB.

It is important that before a designer gets too far in their boot-loader design that they pull out the microcontroller datasheet and read through the flash and memory organization chapters In fact, it is recommend that the designer read it, go to lunch and then read it again. It would be a good idea to read it the next day as well. The primary reason for studying the memory sections of the manual in such detail is that while memory mapping seems straightforward and easy to understand for some reason it always becomes a struggle when implementing the details. It is good for the software developer to get an expert understanding of how the microcontroller memory is organized and how it functions. It won't be long before the developer will need to manipulate the flash memory in order to burn a new application image to it.

The primary purpose for detailed examination of the memory map is to determine what sections of flash are available and best used for the boot-loader. A software developer need to examine the memory map and decide where in flash the boot-loader is going to be located and where in flash the application is located. These two flash areas need to be completely separate from each other.

The boot-loader must understand the existence of the flash memory map so that it can write an application to flash. On the other hand, the application should never know that the boot- loader section exists. That way, there is not the possibility of

the application trying to over-write the boot-loaders program area. This section of flash can usually be protected in a number of ways but in general the two code areas need to be kept separate.

Each microcontroller manufacturer organizes the memory map in a different manner in addition to offering different features that can be utilized by the boot-loader. This can be seen in Figures 2 and 3. Figure 2 shows part of the memory map from a TI TMS320F28035 with 64k of flash broken up into 8 sectors of 8k of flash each. Figure 3 shows part of the memory map from a Free scale S12X with up to 1 MB of flash broken up into 4K sectors.

There are a number of factors that should be taken into account when selecting where to locate the boot-loader.

- Size of the boot-loader
- Location of the vector tables
- Write protection and code security flash sections

Program space is always at a premium. From a manufacturing standpoint the preference is usually to keep the flash size as small as possible in order to save a few cents per unit. This means that we want to keep the boot-loader as small as possible in order to provide the maximum amount of flash space to the application. After all, the developer wants the application to be running and doing all those cool and nifty things not the boot-loader. So how much space should be allocated for the boot-loader?

At a minimum, allocate the smallest erasable block of flash that the processor can handle.

This may be 1k, 4k or even 8k! A good starting rule of thumb is to allocate 4kB of flash space for the boot-loader. Odds are the developer will not come close to filling it but for the first pass allocate 4kB and once everything is working the memory map can be tweaked. The developer should no forget that there is a minimum size that can be erased. For the S12X the minimum flash size that can be erased at a time is 256 bytes. That is easily manageable. However, for the C2000, the minimum

size of flash is an entire 8kB sector!

Another important aspect of the flash to look at is any section specifically designed for a boot-loader? Many processors today have space allocated for the sole purpose of putting a boot- loader there. Special flash protection can often be enabled in these sections so that you can guarantee that the boot-loader won't be erased or over-written by accident. Read that chapter again on memory organization and while you are at it read the one on flash memory. Note anything special about the memory map. For example, the C2000 has One Time Programmable sections and secure code sections while the S12X has only secure code sections. The C2000 OTP section is only 1kB so if the boot-loader is small enough you will want to take advantage of this section. You will not be able to use OTP during development but once the boot-loader works, is tested and validated the linker can be adjusted to use OTP.

Once the flash area(s) has been selected, it is important to update the linker for the application to exclude this area of memory. It is critical to add really good comments on this area of memory and why this range was selected. When maintenance is being performed a year later and someone is looking for more flash space, if the code is documented well the flash space that the boot-loader resides in won't be in jeopardy of being pillaged!

3.8.8 RESET AND INTERRUPT VECTOR

Reset vectors can become a confusing concept when they become mixed in with the development of a boot-loader. When developing application code software developers typically don't need to think about where the reset vector is stored, where it's pointing or really what it is doing. The application libraries just handle everything for us and we end up in main() when we start up our system and start debugging.

The reset vector is the location in memory where the first instruction for the application is located. When a processor is first started up it begins program execution at the address stored in the reset vector. For a system with a boot-loader,

this address will be the branching code or entry into the boot-loader itself. So if the processor reset vector is already used by the boot-loader, how on earth does code branch to the application code? Where is the application reset vector stored?

In simple terms, anywhere the developer wants it to be. One of the most common approaches to handling the reset vector in an application is to leave it in the default location. While in the end this doesn't help, it does for the engineers that are developing the application. If the reset vector is moved to a unique location issues with power-up and resets can result when the boot- loader is not used. Basically the reset vector location that the processor is looking for won't be valid.

The heavy thinking as to where to locate an applications reset vector and how it gets there is squarely placed on the boot-loader designer. The common approach to take is when the binary file is read into the flashing software, it looks through the records and when the application reset vector is found it relocates it to the address specified by the boot-loader. This has the advantage of allowing the application engineer to develop their software without needing to worry about what is going on with the boot-loader (except for flash usage). Another method is to have the boot-loader itself monitor for the reset vector and put it in the location of its choosing. By doing this, the boot- loader designer has complete control over the reset vector and does not need to rely on another application properly translating the reset vector.

It is still up to the boot-loader designer to decide where to place the reset vector. It can really go just about anywhere provided that location is carved out in flash for that purpose. However, the designer will most likely want to select either the first address in the flash space or the last. By doing this contiguous memory space is not interrupted. For example, in the C2000, in the boot-loader you can define a reset vector location in the linker as follows:

APP_RESET : origin = 0x3F5FFB, length = 0x000002

/* Application Reset Vector */

This assumes that sector A is allocated to the boot-loader and the next two addresses after sector A are allocated for the application reset vector. The application

would then start after that.

The reset vector is not the only complication associated with the boot-loader. The boot- loader like any application will need a vector table. The application that it jumps to will also need a vector table. This means that that is an extra vector table that needs to be placed somewhere. In some processors this is not an issue. For example, the C2000 more or less decides where to place it in flash and there is not issue. However, the S12X has a vector table register that needs to be setup to tell the processor where the vectors are located. For the boot-loader it needs to be set to one location while for the application it needs to be set to another.

Typically, the vector table for the application is stored in the flash sector directly following the boot-loader. This allows the application vector table to be at the "top" of flash, where it would typically be found if a boot-loader was not present. The vector table for the boot-loader is stored with the boot-loader in protected flash.

3.8.9 CONCLUSION

While there are 7 classes of requirements for boot-loader design, each of these classes can be broken down to generate a list of requirements that can be used to guide the development of the boot-loader. Below is each of the 7 classes with additional example requirements broken out below them. Each boot-loader will have its own requirements and will be different in certain aspects from another boot-loader.

• Ability to switch or select the operating mode (Application or boot-loader)

a) A GPIO input held for 10 seconds shall cause the system to enter boot-loader mode

b) An EEPROM byte shall be used to switch modes

c) An enter boot-loader message received within 100 ms of start-up shall cause the system to enter boot-loader mode

• Communication interface requirements (USB, CAN, I2C, USART, etc)

a) Application images shall be transferred to the system via UART.

• Record parsing requirement (S-Record, hex, intel, toeff, etc)

a) The application image shall be sent to the system in S-Record format.

• Flash system requirements (erase, write, read, location)

a) The boot-loader shall be capable of erasing only the application section of flash

b) The boot-loader shall write application image records to flash

c) The boot-loader shall be capable of reading back specified address location in flash upon command

d) The boot-loader shall be located in the first 4 kB of flash

• EEPROM requirements (partition, erase, read, write)

a) The boot-loader shall partition EEPROM when commanded

b) The boot-loader shall be capable of erasing the EEPROM

c) The boot-loader shall be capable of reading out a specified EEPROM location

d) The boot-loader shall be capable of writing data to specified EEPROM locations

• Application checksum (verifying the app is not corrupt)

a) The boot-loader shall calculate an image checksum and store it in flash

b) The boot-loader shall verify that the application checksum is valid before executing the application

• Code Security (Protecting the boot-loader and the application)

a) The boot-loader shall be located in protected flash to prevent updating or modification at any time

4

RESULTES & IMPLEMENTATIONS

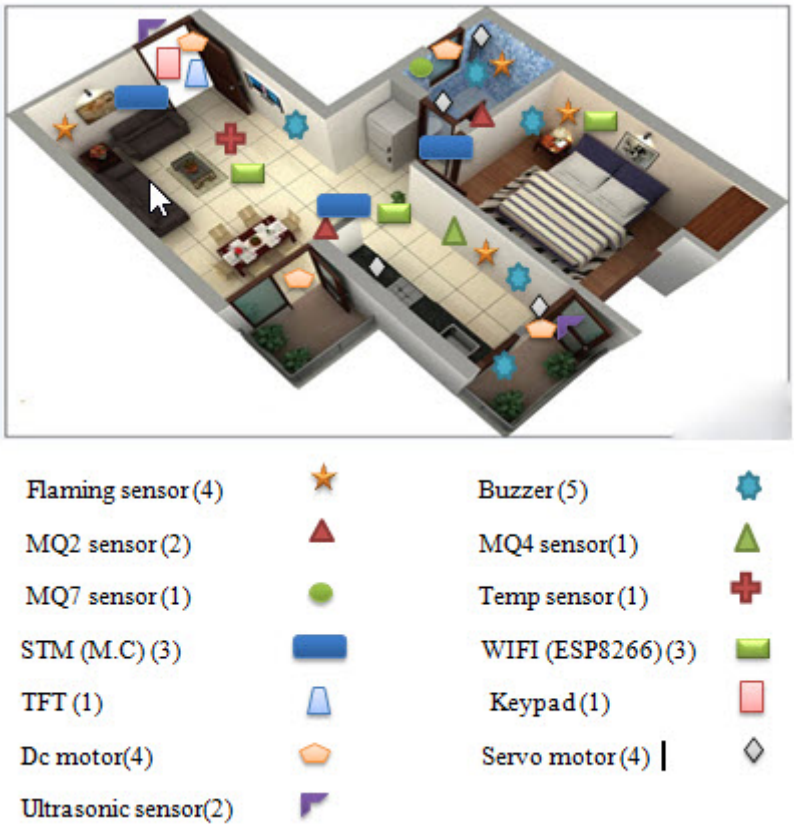

Figure 4.1: The simulated safety and security

As shown in figure 4.1, we design our project in two rooms, kitchen, bathroom and balcony, that make our project consists of five subsystems.

1- Security subsystem.

2- Fire subsystem.

3- Gas subsystem.

4- Iot subsystem.

5- Bootloader subsystem.

we have described every one of them in the previous chapter, in this chapter we will describe the result in every system.

4.1 Security subsystem

This subsystem include five different IOT devices at total nine devices, Some IoT devices are deployed multiple times; for example, Ultrasonic sensor is deployed both

in the door of flat and in the door of balcony .

4.1.1 Components used in this subsystem

We used one Servo Motor that placed on flat door that incorporates positional feedback in order to control the rotational or linear speed and position, it will start to move by angle 90 when the user entered the right password opening the door.

Two Ultrasonic sensors that placed on the first one in the flat door and the second one in the balcony door, to measure the distance or sensing objects are required, the ultrasonic sensor start to sense when it found object at the distance between 5cm to 30cm.

Four Buzzer that placed on the first one in the room1, second one in the room2, third one in the flat door and forth one in the balcony door, When current is applied to the buzzer it causes vibrate. That's the sound that we hear, it start giving voice when the user entered the wrong password four times or the ultrasonic sensor sense object at the distance between 5cm to 30cm.

One TFT Display in the door of flat that used to enhance the operation and usefulness of LCD display (password that user entered right or wrong). 5- One Keypad 4*4 in the door of flat is used to input the password of the door by the user (home owner).

Figure 4.2: Enter Password

Figure 4.3: Right Pass (door is open)

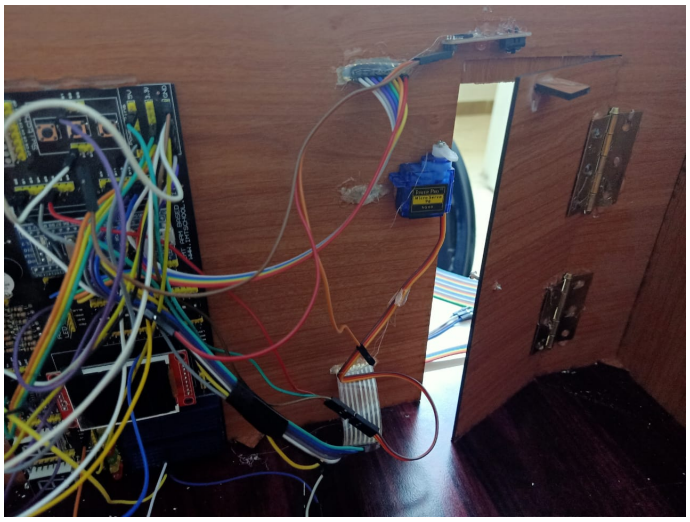

Figure 4.4: The Door is open

Figure 4.5: Wrong Pass (OPSS)

Figure 4.6: Lock

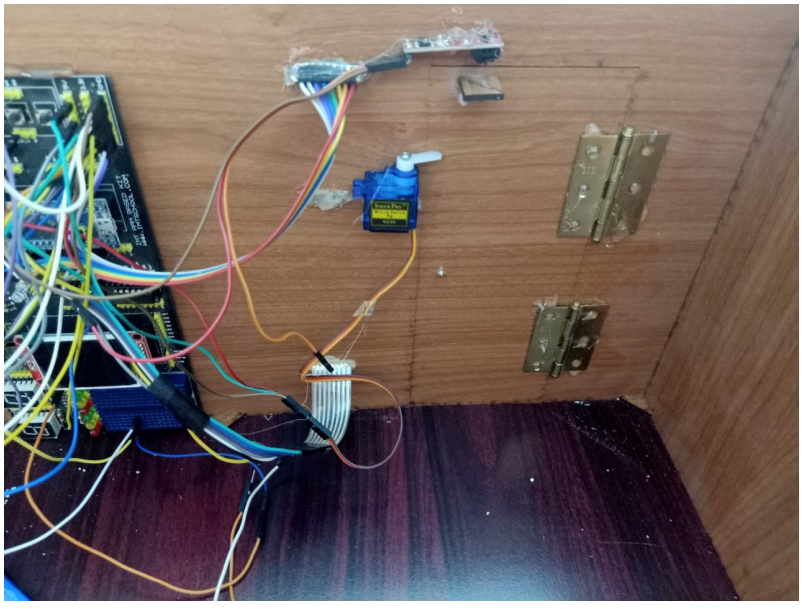

Figure 4.7: The Door is close

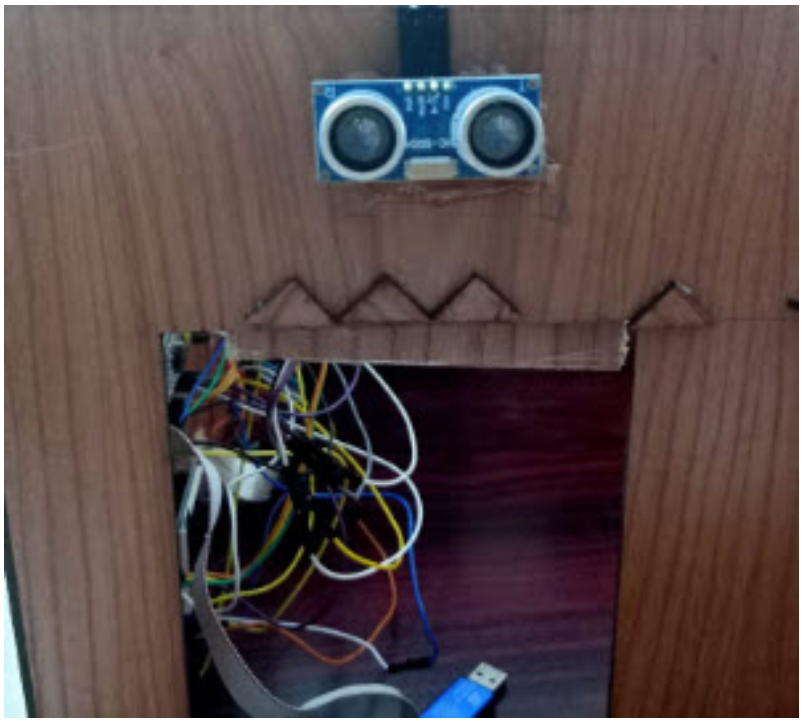

Figure 4.8: Ultrasonic Sensor

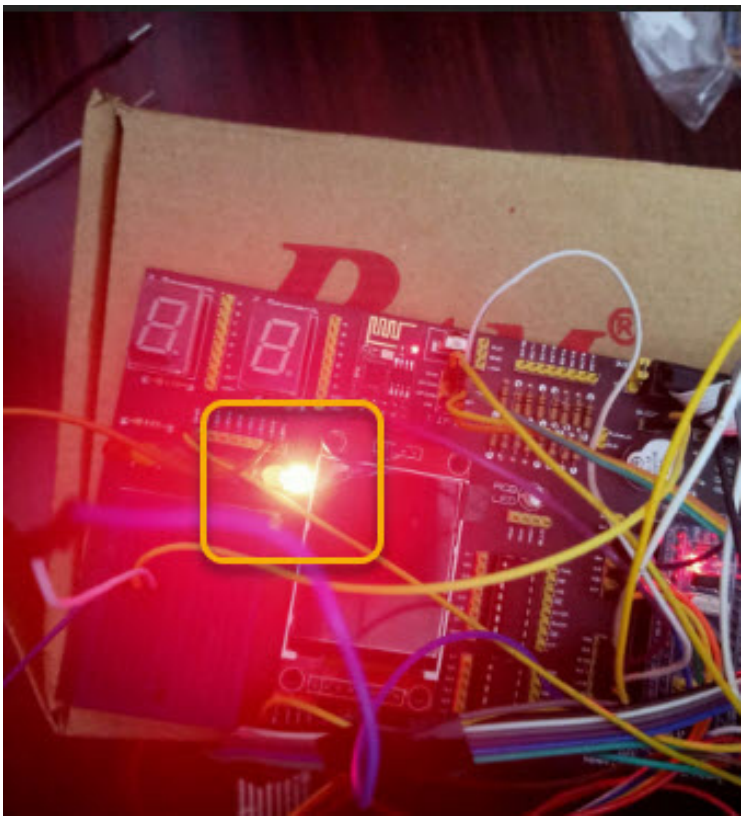

Figure 4.9: Buzzer is turned on

4.2 Fire subsystem

This subsystem include four different IOT devices at total eleven devices, Some IoT devices are deployed multiple times; for example, MQ2 sensor is deployed both in the bathroom and kitchen.

4.2.1 Components used in this subsystem

we used two MQ2 sensor, the first one in the bathroom and second in the kitchen, to detect smoke gas.

Four Flaming sensor, the first one in the bathroom, second in the kitchen, third in room1 and fourth in room2, to detect fire flame.

Four DC motor, the first one in door of the flat, second in window of the flat, third in the door of balcony and fourth in window of kitchen, to move (open) door or window.

Buzzer in bathroom, give sound when detect fire or gas leak.

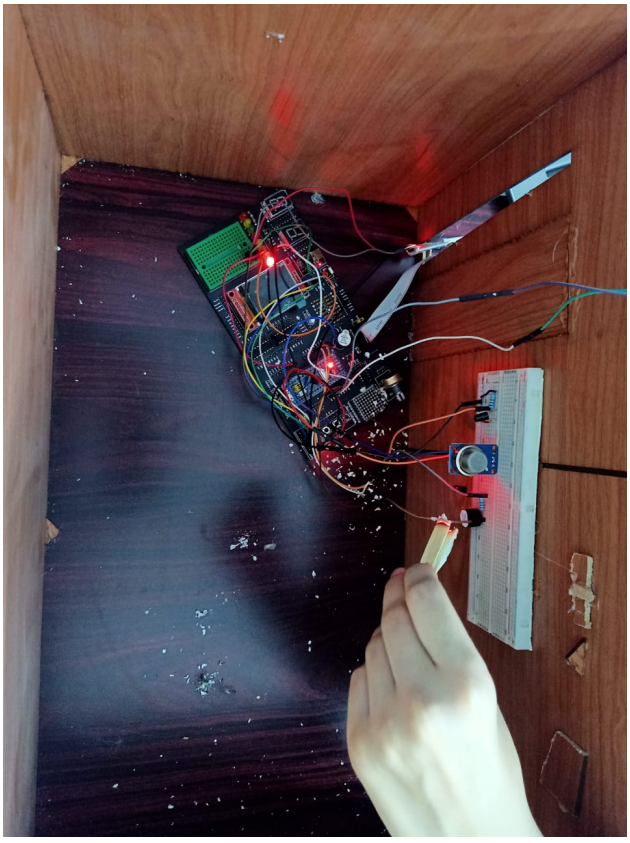

Figure 4.10: Circuit of Fire system

4.3 Gas Subsystem

This subsystem include five different IOT devices, at total seven devices, Some IoT devices are deployed multiple times; for example Servo motor.

4.3.1 Components used in this system

1. MQ4 sensor, in the kitchen to sense natural gas on it.
2. DC Motor, in the kitchen to open window of kitchen
3. Three Servo Motor, two of them in kitchen one to close gas source, second to close general source of gas, one of them in bathroom to close the heater source.
4. Buzzer in kitchen, give sound when detect natural gas leak.
5. Led work with buzzer

Figure 4.11, prove that there is leakage of natural gas, carbon oxide gas in the home, fire in room1 and the home is being robbed.

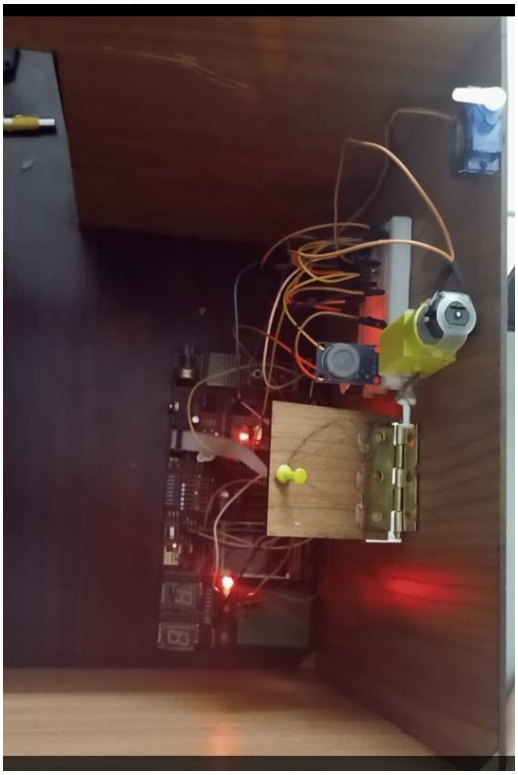

Figure 4.11: Window is open

4.4 IOT subsystem

In this subsystem we used one iot device (ESP8266), two platforms(ThingSpeak and IFTTT) and our server.

1- we used ESP8266 that act as a WIFI connecting servers with each other and with STM32 microcontroller.

2- ThingSpeak platform used to take sensors reads from ESP8266 by TCP protocol

3- TFTTT connected with ThingSpeak platform by URL, when ThingSpeak read high from any sensor then IFTTT send message to user.

4- we created server to user to be able to control his home, turn on/off any sources in his home and know the humidity, gas and temperature degree.

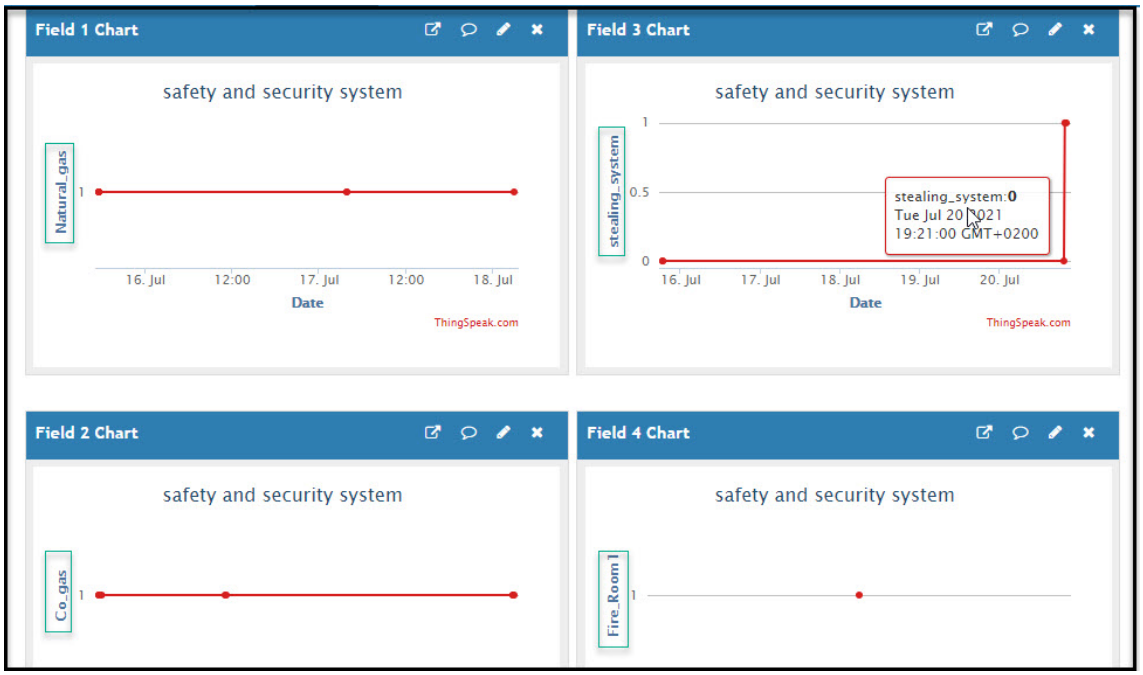

Figure 4.12: ThingSpeak fields of safety & security

Figure 4.13, Indicates the message that the user should receive when his home have fire in room1 and room2.

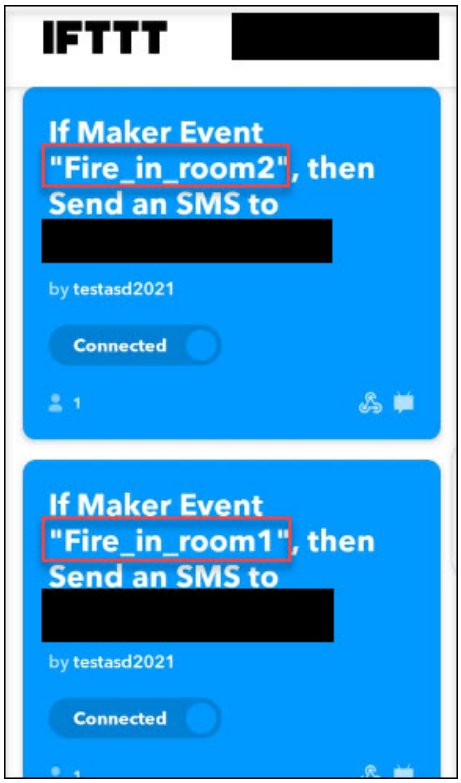

Figure 4.13: Some of IFTTT applets

Figure 4.14, shown the server that work as application for user to control his home

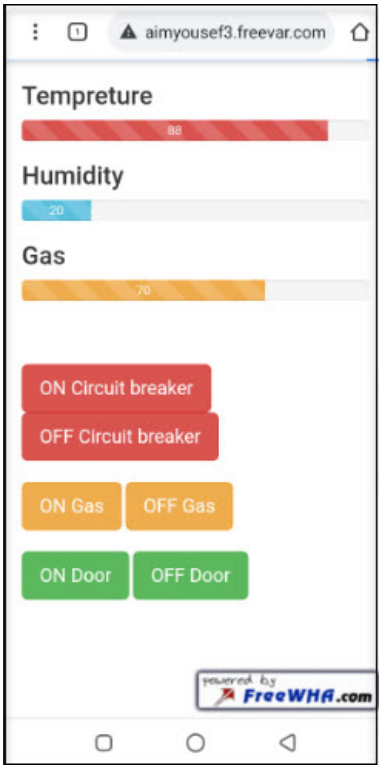

Figure 4.14: user server

5

CONCLUSION

5.1 Conclusions

Connection is the main target of the internet of things. It has been developed over the years to connect things together so that it will make our lives better and easier. The integration of IoT technology and other embedded systems caused revolutionary results in collecting data. As for the trend of big data and analysis of great streams of data, it allowed us to gain better understanding of the world and of ourselves, as well.

Smart devices such as smart houses(safety security), smart watches and smart biomedical devices are very necessary to be taken care of in such a field due to the need to collect and understand the behavior of their users starting from normal people to elderly and patient.

We focused on safety and security on our houses, We gather data about gases, temperature degrees and compare it by normal status so that we may get alerted in cases of emergencies so that we can save people lives and their Properties, House monitoring systems are to help them keep track of any sudden or extreme change in their house and this will also notify their family and loved ones.

5.2 Future Work

More work is required for better safety and security detection algorithms. Power management is a necessity in IoT applications that's why the codes and algorithms used for safety and security recognition and sensors reads shall be optimized. More features shall be added like cameras and microphones for easier communication with the user, as well as, the design of the products will be enhanced.

Refrences

[1] Mohd Azlan Abu, Siti Fatimah Nordin, Mohd Zubir Suboh, Mohd Syazwan Md Yid, and Aizat Faiz Ramli. Design and development of home security systems based on internet of things via favoriot platform. *International Journal of Applied Engineering Research*, 13(2):1253–1260, 2018.

[2] Jacob Beningo. Bootloader design for microcontrollers in embedded systems. In *Embedded Systems Conference, Rev A*, 2015.

[3] AT ESP8266. Instruction set version 0.30. *Espressif Systems IOT Team Copyright (c)*, 2015.

[4] Ibrahim Mohammed Ahmed Farah, Isra'aIbrahim Abdallah, Mohammed Ahmed Mohammed Ahmed, Mohammed Al-Gasim Mohammed Ahmed, et al. *Design and Implementation of an Automatic Fire Fighting System*. PhD thesis, Sudan University of Science and Technology, 2015.

[5] Eleni Isa and Nicolas Sklavos. Smart home automation: Gsm security system design & implementation. *Journal of Engineering Science & Technology Review*, 10(3), 2017.

[6] Mohammad Monirujjaman Khan. Sensor-based gas leakage detector system. In *Engineering Proceedings*, volume 2, page 28. Multidisciplinary Digital Publishing Institute, 2020.

[7] Kalpana Murugan. Intelligent gas booking and leakage system using wireless sensor networks. 2020.

[8] Carmine Noviello. Mastering stm32. *Leadpub. Obtenido de http://www2. keil. com/mdk5/uvision*, 2017.

[9] Juhwan Oh, Zhongwei Jiang, and Henry Panganiban. Development of a smart residential fire protection system. *Advances in Mechanical Engineering*, 5: 825872, 2013.

[10] Arun Raj, Athira Viswanathan, and T Athul. Lpg gas monitoring system. *IJITR*, 3(2):1957–1960, 2015.

[11] Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delahoche, and Christophe Loge. The smart home concept: our immediate future. In *2006 1st IEEE international conference on e-learning in industrial electronics*, pages 23–28. IEEE, 2006.

[12] STM32F102xx STM32F101xx and STM32F105xx STM32F103xx. And stm32f107xx advanced arm-based 32-bit mcus. *STMicroelectronics, RM0008*, 2015.