**Performance Analysis of a concurrent Fire detection and Alarming system**

**M.Tech. Dissertation in**

**Department of Computer Science and Networking**

**Submitted By : Prashaanth R M**

**Reg. No. : 18ETCS037001**

**Supervisors : Jishmi Jos Choondal**

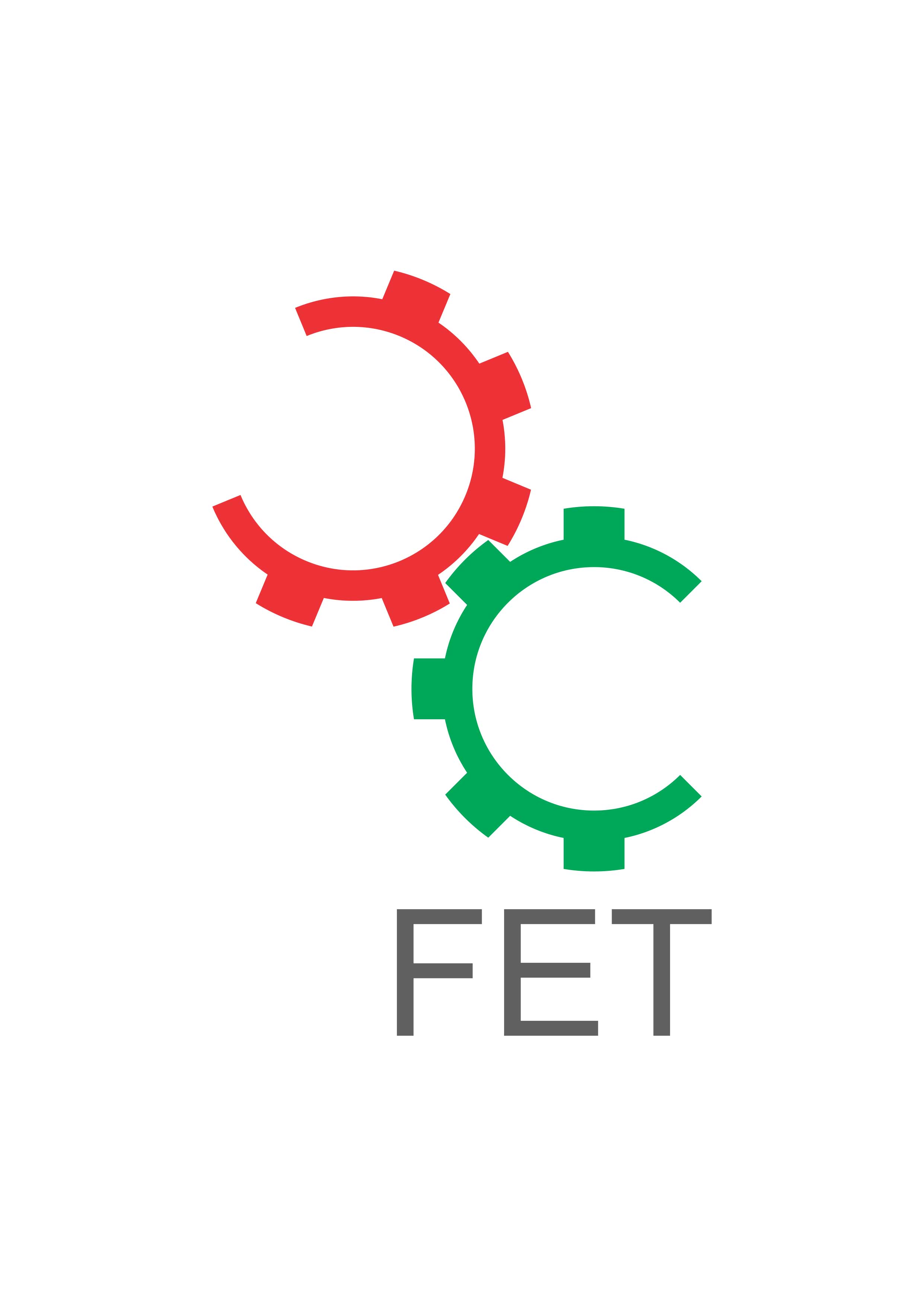
**August – 2020**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**M. S. Ramaiah University of applied sciences**

**Bengaluru -560 054**

**FACULTY OF ENGINEERING AND TECHNOLOGY**



# *Certificate*

*This is to certify that the Dissertation titled* ***“Performance Analysis of a concurrent Fire detection and Alarming system”*** *is a bonafide record of the work carried out by Mr. Prashaanth R M, Reg. No. 18ETCS037001 in partial fulfilment of requirements for the award of M. Tech. Degree of M. S. Ramaiah University of Applied Sciences in the Department of Computer Science and Engineering.*

**August – 2020**

**Asst. Prof. Jishmi Jos Choondal**

Supervisor

**Dr. P. V. R. Murthy Prof. H. M. Rajashekara Swamy**

Head – Dept. of CSE Dean-FET

**Prof. Raghavendra Kulkarni Prof. H. K. Narahari**   
Head–Dept. of CSE Dean-FET

Declaration

**Performance Analysis of a concurrent Fire detection and Alarming System**

The dissertation is submitted in partial fulfilment of academic requirements for the M.Tech. Degree of M. S. Ramaiah University of Applied Sciences in the Department of Computer Science and Engineering. This dissertation is a result of my own investigation. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations, hence this dissertation has been passed through plagiarism check and the report has been submitted to the supervisor.

**Signature :**

**Name of the Student : Mr. Prashaanth R M**

**Reg. No. : 18ETCS037001**

**Date :**

# Acknowledgements

I intend to convey my sincere thanks to my project supervisor Professor. Jishmi Jos Choondal, Asst. Professor, Computer Science and Engineering Department for  
suggesting the dissertation topic and mentoring me throughout the project work.

I would like to thank Dr. P. V. R. Murthy, Head of Computer Science and Engineering Department who has supported through the project with his knowledge and valuable inputs.

I am very obliged to Dr. Sivaguru S Sridharan, Vice Chancellor, Dr. Govind. R. Kadambi, Pro Vice Chancellor and Dr. H. M. Rajasekhara Swamy Dean whose emphasis for quality work helped me keep my focus on my project work and complete it.

I would like to further sincerely thank the manager of M. S. Ramaiah University of Applied Sciences for providing an excellent environment throughout my studies.

Finally, I want to express my honest regards to my beloved parents for their blessings, my family and my fellow classmates for helping me and for their kind wishes for the accomplishment of this project.

Prashaanth R M

18ETCS037001

# Abstract

Technology isn’t in the same plate where it was in 1980’s or 90’s. It has increased in the linear fashion. This is a phase where it is totally filled with Embedded systems. Embedded Systems surround us in the form of gadgets and devices that we use. There is no aspect of human lives, which is untouched by such devices at home or for health diagnostics, transportation, entertainment. Embedded systems are achieved using two things software and hardware. There are many improvements brought in both software as well as the hardware segment. Many advancement in hardware part namely portability, speed of the processor, heat reduction. On the same platter in software side there are some improvements namely different multithreaded programming languages, Real-Time Operating systems, Memory optimizations, optimize execution time, etc.

Embedded systems has brought in a new era called Real Time Embedded systems which has in turn brought in Real Time Applications. Real Time application are being achieved using Real Time threads, threads are nothing but processes. There are many programming languages with multithreading capabilities available like C, C++, Java, Python, Go, Ada, C#, etc. But there are few things to consider here for Real Time Application execution time, memory consumption, compatibility with controllers and other hardware components.

In this thesis a Fire Detection and Alarming system will be designed and developed using both C++ and MicroPython. The system will be developed on ESP32 board using software FreeRTOS, which is compatible with both C++ and Micropython languages. System will be developed in 4 different ways sequentially using C++, Micropython and FreeRTOS using C++, Micropython. Different parameters like execution time, memory consumption, latency, WCRT and Jitter will be examined.

Table of Contents

[*Certificate* II](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\report%20work\final%20report.docx#_Toc51156917)

[Declaration III](#_Toc51156918)

[Acknowledgements IV](#_Toc51156919)

[Abstract V](#_Toc51156920)

[List of figures XI](#_Toc51156921)

[List of tables XIV](#_Toc51156922)

[1. Introduction 16](#_Toc51156923)

[1.1 Introduction 16](#_Toc51156924)

[1.2 Motivation 17](#_Toc51156925)

[1.3 Organization of thesis 18](#_Toc51156926)

[2. Literature Review 20](#_Toc51156927)

[2.1 Background theory 20](#_Toc51156928)

[2.1.1 Multithreading 20](#_Toc51156929)

[2.1.1.1 Advantages of Multithreading applications 20](#_Toc51156930)

[2.1.1.2 Disadvantages of Multithreading application: 21](#_Toc51156931)

[2.1.2 Different operating systems 21](#_Toc51156932)

[2.1.2.1 General Purpose operating system 22](#_Toc51156933)

[2.1.2.2 Real Time operating system 22](#_Toc51156934)

[2.1.2.3 RTOS vs GPOS 24](#_Toc51156935)

[2.1.3 FreeRTOS 24](#_Toc51156936)

[2.1.3.1 Implementation of FreeRTOS 25](#_Toc51156937)

[2.1.4 Micropython 27](#_Toc51156938)

[2.1.4.1 Compatibility of Micropython with microcontrollers: 27](#_Toc51156939)

[2.1.5 Programming languages compatibility with microcontrollers 28](#_Toc51156940)

[2.1.6 Selection of Real Time Application 31](#_Toc51156941)

[2.2 Literature Review 32](#_Toc51156942)

[2.2.1 Overview of Papers and Journals Referred 32](#_Toc51156943)

[2.2.2 Literature Summary 39](#_Toc51156944)

[2.3 Problem Formulation 49](#_Toc51156945)

[2.3.1 Identification of Research Gaps 49](#_Toc51156946)

[2.3.2 Research questions the dissertation would like to address 49](#_Toc51156947)

[3. Problem Statement 50](#_Toc51156948)

[3.1 Title 50](#_Toc51156949)

[3.2 Aim 50](#_Toc51156950)

[3.3 Project Objectives 50](#_Toc51156951)

[3.4 Methods and Methodology 51](#_Toc51156952)

[4. Design and Implementation 53](#_Toc51156953)

[4.1 Requirement analysis 53](#_Toc51156954)

[4.1.1 Functional Requirements 53](#_Toc51156955)

[4.1.2 Non-functional Requirements 53](#_Toc51156956)

[4.2 System Specifications 54](#_Toc51156957)

[4.2.1 Hardware requirements 54](#_Toc51156958)

[4.2.1.1 Embedded boards: 54](#_Toc51156959)

[4.2.1.2 Sensors: 57](#_Toc51156960)

[4.2.1.3 Actuators: 60](#_Toc51156961)

[4.2.1.4 Laptop: 64](#_Toc51156962)

[4.2.2 Software requirements 64](#_Toc51156963)

[4.2.2.1 Arduino IDE: 64](#_Toc51156964)

[4.2.2.2 Visual Studio Code: 65](#_Toc51156965)

[4.2.3 Selected System components and Development costing 66](#_Toc51156966)

[4.3 Design Specification 67](#_Toc51156967)

[4.4 System Design 67](#_Toc51156968)

[4.4.1 Block diagram of system 67](#_Toc51156969)

[4.4.2 Schematic Diagram 69](#_Toc51156970)

[4.4.3 Flow chart of the system 70](#_Toc51156971)

[4.5 Implementation of System 72](#_Toc51156972)

[4.5.1 Hardware Setup: 72](#_Toc51156973)

[4.5.2 Software Setup: 73](#_Toc51156974)

[4.5.2.1 Setting up ESP32 in Arduino IDE: 73](#_Toc51156975)

[4.5.2.2 Setting up Micropython to ESP32 board 77](#_Toc51156976)

[4.5.2.3 Setting up Visual Studio Code for MicroPython 81](#_Toc51156977)

[4.5.3 System Implementation 83](#_Toc51156978)

[4.5.3.1 Implementation between ESP32 and sequential C++ programming 83](#_Toc51156979)

[4.5.3.2 Implementation between ESP32 and multithreaded C++ programming 84](#_Toc51156980)

[4.5.3.3 Implementation between ESP32 and sequential Micropython 86](#_Toc51156981)

[4.5.3.4 Implementation between ESP32 and Micropython with threads 88](#_Toc51156982)

[4.6 System Integration: 90](#_Toc51156983)

[5. Results and Discussions 91](#_Toc51156984)

[5.1 Unit Testing 91](#_Toc51156985)

[5.2 Integration Testing 97](#_Toc51156986)

[5.3 System level Testing 99](#_Toc51156987)

[5.4 Result Analysis 101](#_Toc51156988)

[5.4.1 Performance Analysis: 103](#_Toc51156989)

[5.4.1.1 Latency: 103](#_Toc51156990)

[5.4.1.2 Worst case response time: 106](#_Toc51156991)

[5.4.1.3 Jitter: 108](#_Toc51156992)

[5.4.1.4 Execution time: 109](#_Toc51156993)

[5.4.1.5 Allocated heap memory: 110](#_Toc51156994)

[5.4.2 Existing system vs Proposed system 111](#_Toc51156995)

[6. Conclusions and Future Directions 113](#_Toc51156996)

[6.1 Summary of Work 113](#_Toc51156997)

[6.2 Conclusion 113](#_Toc51156998)

[6.3 Future Work 114](#_Toc51156999)

[References 114](#_Toc51157000)

# List of figures

[Figure 2. 1: RTOS architecture 20](#_Toc50313441)

[Figure 2. 2: FreeRTOS architecture 23](#_Toc50313442)

[Figure 4. 1: ESP32 Dev module 55](#_Toc50313557)

[Figure 4. 2: Flame sensor YS-17 56](#_Toc50313558)

[Figure 4. 3: Water level sensor 57](#_Toc50313559)

[Figure 4. 4: Smoke sensor MQ-2 58](#_Toc50313560)

[Figure 4. 5: DC motor 59](#_Toc50313561)

[Figure 4. 6: Light Emitting Diode 60](#_Toc50313562)

[Figure 4. 7: Buzzer 61](#_Toc50313563)

[Figure 4. 8: Submersible water pump 61](#_Toc50313564)

[Figure 4. 9: Arduino IDE UI 63](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313565)

[Figure 4. 10: High level Block diagram 66](#_Toc50313566)

[Figure 4. 11: Low Level Block Diagram 67](#_Toc50313567)

[Figure 4. 12: Schematic Diagram 68](#_Toc50313568)

[Figure 4. 13: Flow chart 69](#_Toc50313569)

[Figure 4. 14: Hardware setup 70](#_Toc50313570)

[Figure 4. 15 Hardware setup of System developed 70](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313571)

[Figure 4. 16: Preference Dialog box 72](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313572)

[Figure 4. 17: ESP32 data package 73](#_Toc50313573)

[Figure 4. 18: Board manager Dialog 74](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313574)

[Figure 4. 19: configuring COM port Speed 74](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313575)

[Figure 4. 20: Testing the working 75](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313576)

[Figure 4. 21: Installing python 76](file:///C:\Users\prashanth\Desktop\msruas\Dissertation%20multithreading\final%20report.docx#_Toc50313577)

[Figure 4. 22: Checking Python's working 76](#_Toc50313578)

[Figure 4. 23: Install esptool 77](#_Toc50313579)

[Figure 4. 24: Erase flash memory 77](#_Toc50313580)

[Figure 4. 25: Dump the Micropython firmware 78](#_Toc50313581)

[Figure 4. 26: Testing Micropython 79](#_Toc50313582)

[Figure 4. 27: Pymakr extension 80](#_Toc50313583)

[Figure 4. 28: Detecting COM port 80](#_Toc50313584)

[Figure 4. 29: Sequential C++ Code snippet 1 81](#_Toc50313585)

[Figure 4. 30: Sequential C++ code snippet 2 82](#_Toc50313586)

[Figure 4. 31: Multithreaded C++ code snippet 1 83](#_Toc50313587)

[Figure 4. 32: Multithreaded C++ code snippet 2 84](#_Toc50313588)

[Figure 4. 33: Sequential Micropython code snippet 1 85](#_Toc50313589)

[Figure 4. 34: Sequential Micropython code snippet 2 86](#_Toc50313590)

[Figure 4. 35: Multithreaded micropython code snippet 1 87](#_Toc50313591)

[Figure 4. 36: Multithreaded Micropython code snippet 2 87](#_Toc50313592)

[Figure 4. 37: Latency test setup 88](#_Toc50313593)

[Figure 5.1: Smoke sensor interface with ESP32 93](#_Toc50313658)

[Figure 5. 2: Flame sensor integration with ESP32 94](#_Toc50313659)

[Figure 5. 3: Water level sensor integration with ESP32 95](#_Toc50313660)

[Figure 5. 4: DC motor integration with ESP32 95](#_Toc50313661)

[Figure 5. 5: Submersible water pump interface with ESP32 96](#_Toc50313662)

[Figure 5. 6: Default result monitor screen for Micropython 100](#_Toc50313663)

[Figure 5. 7: Result monitor screen after manual interrupt is pressed in Micropython 100](#_Toc50313664)

[Figure 5. 8: Default output screen C++ 101](#_Toc50313665)

[Figure 5. 9: Manual press triggered actuation output screen C++ 101](#_Toc50313666)

[Figure 5. 10: Minimal load latency test 102](#_Toc50313667)

[Figure 5. 11: Sequential C++ CRO screen 103](#_Toc50313668)

[Figure 5. 12: Maximum load for latency test 104](#_Toc50313669)

[Figure 5. 13: Multithreaded C++ CRO screen 105](#_Toc50313670)

[Figure 5. 14: Minimal load WCRT 106](#_Toc50313671)

[Figure 5. 15: Maximum load WCRT 107](#_Toc50313672)

[Figure 5. 16: Jitter 108](#_Toc50313673)

[Figure 5. 17: Execution time 109](#_Toc50313674)

Figure 5.18: Allocated Heap Memory……………………………………………………………………………………. 111  
Figure 5.19: Existing system vs Proposed system……………………………………………………………………112

# 

# List of tables

[Table 2. 1: RTOS vs GPOS 23](#_Toc50316933)

[Table 2. 2: Hardware compatibility analysis 28](#_Toc50316934)

[Table 2. 3: Hardware compatibility with programming languages 29](#_Toc50316935)

[Table 2. 4: Literature summary for referred papers 38](#_Toc50316936)

[Table 3. 1: Methods and Methodology to achieve defined objectives 50](#_Toc50316941)

[Table 4. 1: Embedded boards comparison 53](#_Toc50316948)

[Table 4. 2: Hardware and software costs 65](#_Toc50316949)

[Table 5. 1: Unit testing test cases 90](#_Toc50316964)

[Table 5. 2: Integration testing test cases 97](#_Toc50316965)

[Table 5. 3: System level testing test cases 98](#_Toc50316966)

# 1. Introduction

## 1.1 Introduction

The current generation is being filled with variety of Embedded systems. Starting from your Television, mobile phones, washing machine etc all are an embedded system. The concept of embedded system was formed in 1965 from the Computer there is where everything started. These systems have reduced two things one is manual work and another is the time taken for completion. Slowly when comparing the old computer and the one today there are a lot more differences like size, features, speed etc. This major drift has been carrying out speed all over the years. Speed is being achieved using powerful hardware and optimized and well-written firmware.

There has been improvement from single-core to multi-core processors in the architecture of the hardware systems. Not only that even mobile phones are like a mini computer today which is achieved using Octo-core processors. Major drift in speed is majorly concentrated since due to fast life people do not have that much time for wasting in opening different applications in a computer or a mobile phone or it may be any embedded device. To a greater extent improvisation on hardware has made software to upgrade concurrently. Strictly bringing in many features to the system has brought in many interrupt that is being solved using multithreaded programming.

Multithreaded programming contains two or more threads that are running concurrently and always looking for a time-saving notion. An interrupt will react like a process and a corresponding scheduler will be assigned to perform it. This gave birth to real-time applications which are time deterministic and the system should complete the corresponding task within the specified given time. Real-time application is being used in Automotive, aerospace, military, gaming, Server creation, space shuttle etc. When the real-time application came up it was challenging to achieve the time deterministic nature using a general-purpose operating system so, for this, a Real-time operating system was required. The first operating system was launched way back when Apollo 11 shuttle was made and it was the AGC computer weighed in at 30 kg, ran at 1MHz and only had 74Kb of ROM and 4Kb of RAM but now we have many RTOS like Xenomai, VxWorks, FreeRTOS, OSEK, QNX, LynxOS etc.

There are many differences between RTOS and GPOS right from their scheduling, priority, threads, memory utilization etc. General purpose operating systems cannot perform real time tasks whereas RTOS is suitable for real time applications. Latency is a problem with GPOS but it is overcome using real time OS. Priority inversion cannot be done in GPOS, it is done with real time kernel. Different programming languages have different capabilities and there are many languages like C, Java, C++, python etc that has multithreading capabilities and the real-time capabilities are added in different ways in each programming language. The real-time application developed using different languages like C++ and Python will behave in a different way which will be analyzed here.

## 1.2 Motivation

Embedded system has a wide range of study. Every day we come across hundreds to thousands Embedded system. The current generation that is followed in Embedded systems is being filled with different real-time applications it may be hard, soft or firm real-time applications. Real-time applications are very time-critical and deadline-driven which are achieved using real-time threads. Multithreaded programming drives the RTA in and out that in turn motivates the user to use that particular application.

Many languages have multithreading support but there are very few languages which are being used in the market for developing. So, the C++ (POSIX threads) and Python language will be taken as the first preference and their real-time capabilities will be studied by developing a real-time application. Parameters like Latency, Jitter, WCRT, Execution time, Memory consumption and Correctness will be tested. Based on these analysis it can be easy to examine the two different languages capabilities in developing a real-time application.

## 1.3 Organization of thesis

The dissertation has six chapters which include an introduction to the topic, background theory and literature survey, problem statement, design and implementation, results and discussion and conclusion. The first part of dissertation includes brief about the history of Embedded systems, how did it evolve, Multithreaded programming, the difference between GPOS and RTOS and how different programming languages can be used to develop a real time application. The second part comprises details descriptions about Micropython, RTOS vs GPOS, FreeRTOS and sensors used in the application. In third part of thesis, the functionality of work and expected results that has been implemented is presented. Rest of the parts includes design specifications, implementation, and remarks for obtained results and reviews the work with some proposals for how to expand the work done by additional development. The organization is briefly listed below,

Chapter 1: It gives introduction about multithreaded programming and Real Time applications to examine the complexities in C++ and Python programming languages.

Chapter 2: In this chapter, Python for Embedded systems which is Micropython will be discussed. Different programming languages and their compatibility with microcontrollers and hardware implementation features will be explained.

Chapter 3: This chapter gives important information about dissertation like, title of project, aim of the project. For more to that major objectives of work also listed followed by listed methods to achieve objectives.

Chapter 4: Main part of the thesis discussed in this chapter. Requirements analysis for the work with design constraints like, block diagram, flow chart and algorithms etc. explained and written in this.

Chapter 5: After implementation stage, results of the work presented along with relevant discussion based on the results found in this chapter.

Chapter 6: Last chapter of the thesis which includes the summary and conclusion. In addition to that future expansion also discussed for the developed system.

# 2. Literature Review

The chapter comprises of three topics, background theory and the overview of the papers, literature gaps identified and the summary of the papers referred. Technical papers and documents which are relevant to the topic are considered and reviewed.

## 2.1 Background theory

This section includes the background study on the topics of Multithreading, Micropython, different programming languages compatibility with microcontrollers and RTOS vs GPOS will be discussed.

### 2.1.1 Multithreading

Multithreading has been in boom since 1995. When there are two or more threads performing a process concurrently in a Central Processing Unit is called as multithreading. This totally differs from multiprocessing. Multiprocessor system are totally different because they are 2 or more cores itself sharing the work. But, multithreading is 2 or more threads in a single processor (i.e) core. Each thread has its own stack, thread ID, program counter/stack pointer, priority and thread private storage.

#### 2.1.1.1 Advantages of Multithreading applications

**Responsiveness:** Multithreading can allow an application to remain responsive to input. In a one-thread program, if the main execution thread blocks on a long-running task, the entire application can appear to freeze. By moving such long-running tasks to a *worker thread* that runs concurrently with the main execution thread, it is possible for the application to remain responsive to user input while executing tasks in the background.

**Faster Execution:** This advantage of a multithreaded program allows it to operate faster on computer systems that have multiple central processing units (CPUs) or one or more multi-core processors, or across a cluster of machines, because the threads of the program naturally lend themselves to parallel execution, assuming sufficient independence.

**Lower resource consumption:** Using threads, an application can serve multiple clients concurrently using fewer resources than it would need when using multiple process copies of itself. For example, the Apache HTTP server uses thread pools: a pool of listener threads for listening to incoming requests, and a pool of server threads for processing those requests.

**Better system utilization:** As an example, a file system using multiple threads can achieve higher throughput and lower latency since data in a faster medium (such as cache memory) can be retrieved by one thread while another thread retrieves data from a slower medium (such as external storage) with neither thread waiting for the other to finish.

**Simplified sharing and communication:** Unlike processes, which require a message passing or shared memory mechanism to perform inter-process communication (IPC), threads can communicate through data, code and files they already share.

#### 2.1.1.2 Disadvantages of Multithreading application:

**Synchronization:** Since threads share the same address space, the programmer must be careful to avoid race conditions and other non-intuitive behaviors. In order for data to be correctly manipulated, threads will often need to rendezvous in time in order to process the data in the correct order. Threads may also require mutually exclusive operations (often implemented using mutexes) to prevent common data from being read or overwritten in one thread while being modified by another. Careless use of such primitives can lead to deadlocks, livelocks or races over resources.

**Thread crashes a process:** An illegal operation performed by a thread crashes the entire process; therefore, one misbehaving thread can disrupt the processing of all the other threads in the application.

### 2.1.2 Different operating systems

The operating system is the one which guides the hardware and schedules the process. Operating system lies on the top of the microcontroller. While General Purpose Operating Systems (GPOS) can handle multiple tasks efficiently, they usually do so without the pressure of time running out. RTOS on the other hand is designed to deliver an accurate output within the expected timeline (which as stated earlier, is akin to the time taken for the blink of an eye).

#### 2.1.2.1 General Purpose operating system

General Purpose operating system takes care of many tasks but it’s not time deterministic. The process takes huge amount of time to complete. The memory management function keeps track of the status of each memory location, either allocated or free. It determines how memory is allocated among competing processes, deciding which gets memory, when they receive it, and how much they are allowed. An operating system must also keep track of programs in memory.

The applications provide the functionality that the user of the computer wants or needs. The services provided by the operating system make writing the applications faster, simpler, and more maintainable. If you are reading this web page, then you are using a web browser (the application program that provides the functionality you are interested in), which will itself be running in an environment provided by an operating system.

#### 2.1.2.2 Real Time operating system

Real Time Operating System (RTOS) on hearing this term many of the programmers and computer enthusiasts think differently. It differs from the general purpose OS, RTOS is a real time OS working with real time constraints as power, time and efficient usage of memory. Most of the embedded systems are bound to real time constraints and it is achieved using real time system. General purpose operating systems are suitable to do multiple tasks at the same time, but synchronization is a problem with GPOS. To do multiple tasks with worst case execution time on a particular architecture real time OS is used in an embedded computing system. The main concern of RTOS is it produces an accurate output within the deadline or time.

The behaviour of an embedded system or general purpose machine depends upon the nature of application design. A general purpose operating system was designed to handle multiple tasks with no time limit, we cannot say in a certain time a task will get happened.

An example is form your PC if you copy some data from one device to another device it may take several minutes or more, we cannot predict responsiveness of the system of course, due to the tasks running parallel at that time.

But, a real time embedded system can give an accurate output at right time that means, it is time critical no delay is encouraged for real time systems and if any delay occurs it may lead to catastrophic effects. For example Airbag control system in a car. If you drive a car at a high speed accidents may happen, in such case airbag opens and saves your life.

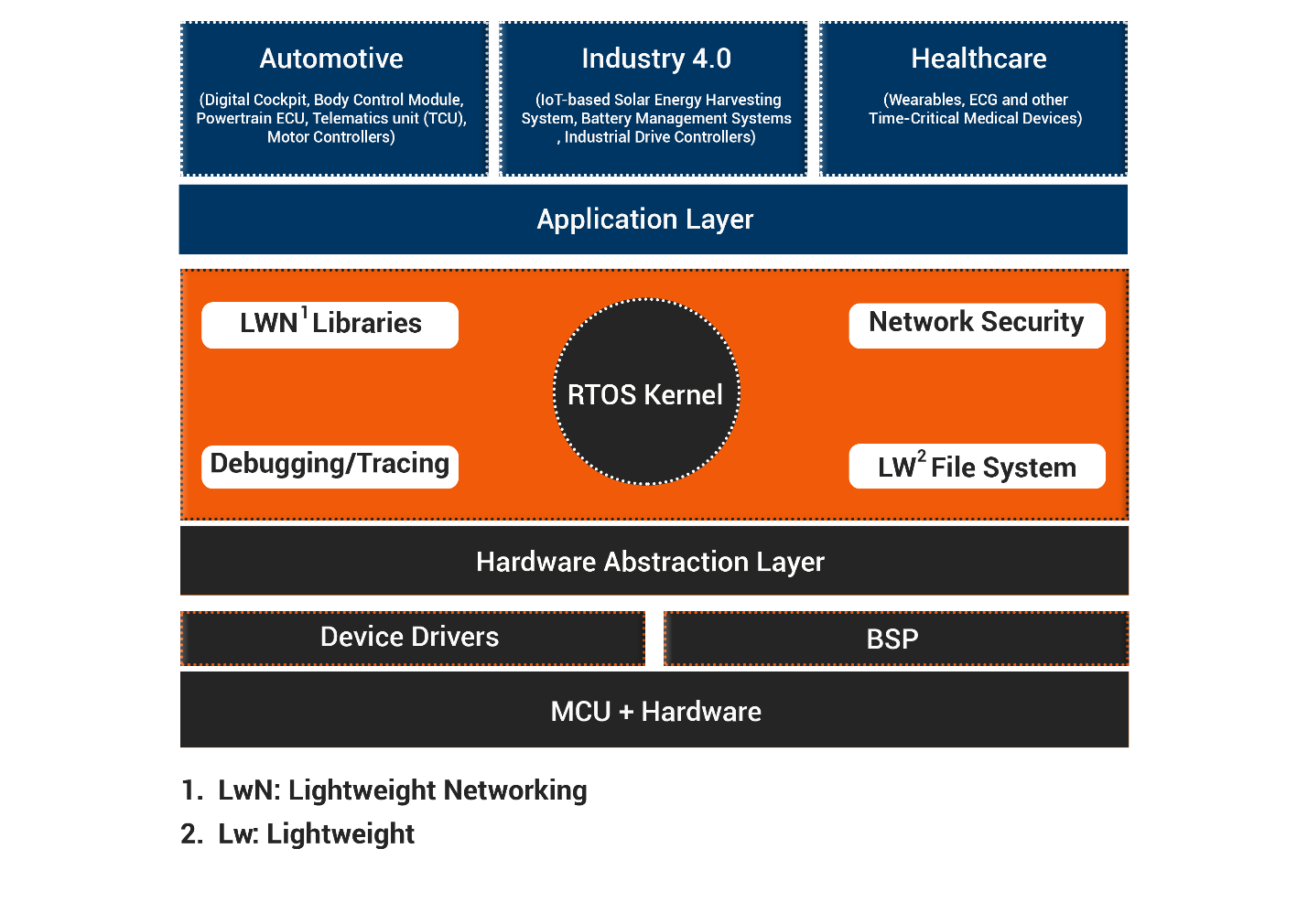


Figure 2. 1: RTOS architecture

#### 2.1.2.3 RTOS vs GPOS

Table 2. 1: RTOS vs GPOS

|  |  |
| --- | --- |
| GPOS | RTOS |
| * Dynamic memory mapping * Random Execution Pattern * Response Times not Guaranteed * Examples: Microsoft Windows operating system, Apple macOS operating system, Red Hat Enterprise Linux operating system | * Deterministic: no random execution pattern * Predictable Response Times * Time Bound * Preemptive Kernel * Examples: FreeRTOS, Xenomai, QNX, VxWorks. |

### 2.1.3 FreeRTOS

FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller – although its use is not limited to microcontroller applications.

A microcontroller is a small and resource constrained processor that incorporates, on a single chip, the processor itself, read only memory (ROM or Flash) to hold the program to be executed, and the random access memory (RAM) needed by the programs it executes. Typically the program is executed directly from the read only memory.

Microcontrollers are used in deeply embedded applications (those applications where you never actually see the processors themselves, or the software they are running) that normally have a very specific and dedicated job to do. The size constraints, and dedicated end application nature, rarely warrant the use of a full RTOS implementation – or indeed make the use of a full RTOS implementation possible. FreeRTOS therefore provides the core real time scheduling functionality, inter-task communication, timing and synchronisation primitives only. This means it is more accurately described as a real time kernel, or real time executive. Additional functionality, such as a command console interface, or networking stacks, can then be included with add-on components.

#### 2.1.3.1 Implementation of FreeRTOS

FreeRTOS is designed to be small and simple. The kernel itself consists of only three C files. To make the code readable, easy to port, and maintainable, it is written mostly in [C](https://en.wikipedia.org/wiki/C_(programming_language)), but there are a few assembly functions included where needed (mostly in architecture-specific scheduler routines).

FreeRTOS provides methods for multiple threads or tasks, mutexes, semaphores and software timers. A tick-less mode is provided for low power applications. Thread priorities are supported. FreeRTOS applications can be completely statically allocated. Alternatively RTOS objects can be dynamically allocated with five schemes of memory allocation provided:

* Allocate only;
* Allocate and free with a very simple, fast, algorithm;
* A more complex but fast allocate and free algorithm with memory coalescence;
* An alternative to the more complex scheme that includes memory coalescence that allows a heap to be broken across multiple memory areas and,
* C library allocate and free with some mutual exclusion protection.

There are none of the more advanced features typically found in operating systems like Linux or Microsoft Windows, such as device drivers, advanced memory management, user accounts, and networking. The emphasis is on compactness and speed of execution. FreeRTOS can be thought of as a 'thread library' rather than an 'operating system', although command line interface and POSIX-like I/O abstraction add-ons are available.

FreeRTOS implements multiple threads by having the host program call a thread tick method at regular short intervals. The thread tick method switches tasks depending on priority and a round-robin scheduling scheme. The usual interval is 1/1000 of a second to 1/100 of a second, via an interrupt from a hardware timer, but this interval is often changed to suit a particular application.

The download contains prepared configurations and demonstrations for every port and compiler, allowing rapid application design. The FreeRTOS.org site also contains a lot of documentation and RTOS tutorials (additional manuals and tutorials are available for a fee), as well as details of the RTOS design.

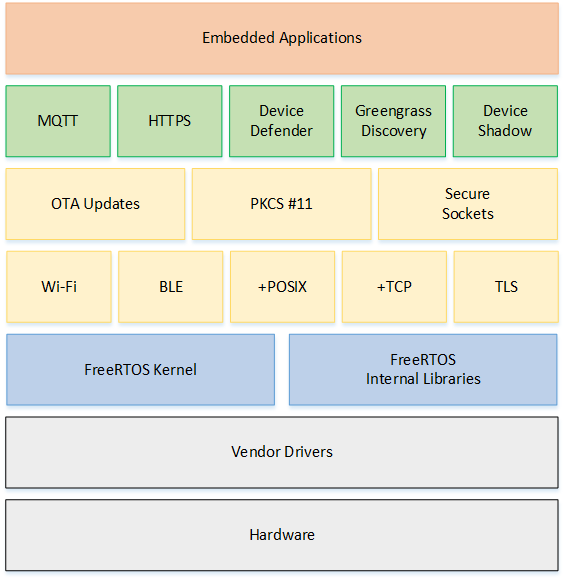


Figure 2. 2: FreeRTOS architecture

### 2.1.4 Micropython

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers and in constrained environments.

MicroPython is packed full of advanced features such as an interactive prompt, arbitrary precision integers, closures, list comprehension, generators, exception handling and more. Yet it is compact enough to fit and run within just 256k of code space and 16k of RAM. MicroPython aims to be as compatible with normal Python as possible to allow you to transfer code with ease from the desktop to a microcontroller or embedded system.

MicroPython is a full Python compiler and runtime that runs on the bare-metal. You get an interactive prompt (the REPL) to execute commands immediately, along with the ability to run and import scripts from the built-in filesystem. The REPL has history, tab completion, auto-indent and paste mode for a great user experience.

MicroPython strives to be as compatible as possible with normal Python (known as CPython) so that if you know Python you already know MicroPython. On the other hand, the more you learn about MicroPython the better you become at Python. In addition to implementing a selection of core Python libraries, MicroPython includes modules such as "machine" for accessing low-level hardware.

MicroPython includes a cross compiler which generates MicroPython bytecode (file extension .mpy). The Python code can be compiled into the bytecode either directly on a microcontroller or it can be precompiled elsewhere. MicroPython firmware can be built without the compiler, leaving only the virtual machine which can run the precompiled mpy programs.

#### 2.1.4.1 Compatibility of Micropython with microcontrollers:

Micropython is compatible with the following microcontrollers,

* STM32
* Pyboard
* ESP32
* ESP8266
* Arduino Due

Also, micropython is compatible with microcontroller which has ARM processor as their architecture. In this thesis Micropython will be used on ESP32 and the output will be analysed. Implementation of Micropython is shown in Chapter 4.

### 2.1.5 Programming languages compatibility with microcontrollers

If there is only software part for the project there is no need for any confusions. But there is a controller to be chosen for working with different sensors and actuators which is to be interfaced with it. So, there are few things that are to be considered like,

* First the microcontroller should accept the language.
* Secondly the microcontroller should have RTOS capabilities (preferably FreeRTOS).

Not all programming languages are compatible with all microcontrollers. Our goal was to use the same design, RTOS and microcontroller for programming both C++ as well as Micropython. The microprocessor and microcontrollers available in the market are as follows,

* Beagle bone
* Raspberry Pi
* ESP32
* Arduino
* STM32

Our main goal was to find what RTOS and microcontroller platform was compatible with all 3 languages C++, Python and Java.

Table 2. 2: Hardware compatibility analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Hardware** | **RTOS/platform** | **Language** | **Thoughts** |
| Beagle Bone Black | Xenomai patch is already just use it for RTOS capabilities | C/C++, Python can also be done. | No information was available for Java. Microprocessors are not better for Real-Time application. |
|  |  |  |  |
| Raspberry Pi | But FreeRTOS can’t be ported. | Just for C program interfacing with sensors. Normal sequential program with Java is possible | Microprocessors are not better for Real-Time application. |
| ARDUINO with Embedded C++ | FREERTOS [Not a full- fledged OS] | C++ programming | It is possible |
| ARDUINO with Java | [Java SE is used](https://www.youtube.com/watch?v=MsWIVsXR8bk) | [Java coding](file:///C:\Users\jishmi\Downloads\•%09https:\www.youtube.com\watch%3fv=5N30jHMhw9c) | It is possible but tried to work this out but there are some errors coming up. |
| Arduino with Python | Possible Arduino Due board is required since micropython works with only ARM processors. | MicroPython | Possible but only sequential programming can be done. |
| [Arduino with Python](https://micropython.org/download/) | Freertos | [Python](https://github.com/thearn/Python-Arduino-Command-API) | Not Possible |
| ESP32 (I have experience) with Embedded C++ | FREERTOS [Not a full- fledged OS] | C++ coding | Possible with RTOS |
| Esp32 with python | Micropython | Python 3 | Possible with RTOS |
| Esp32 with Java | Jamaica VM a real time java virtual machine. | Java | Possible with only ESP32 Wrover board |

So, the decision from here was straight forward. Decision was made to do the proceedings with C++ and Python since, Java’s compatibility wasn’t proved anywhere.

Table 2. 3: Hardware compatibility with programming languages

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hardware platform** | **C/C++** | **Java** | **Python** | **Recommended** |
| Raspberry Pi | Yes(NO RTOS) | Yes (No RTOS) | Yes(no RTOS) | No |
| Beagle Bone Black | Yes | Yes (No RTOS) | Yes(No RTOS) | No |
| ESP32 | Yes(RTOS possible) | [Yes (No RTOS)with ESP wrover board](https://developer.microej.com/get-started-app-development-on-espressif-esp32-wroverkit-v41/) | [Yes(Possible)](https://docs.micropython.org/en/latest/esp32/tutorial/intro.html) | Yes (From here RTOS used was FreeRTOS and board was ESP32 Dev module). The same will be used for programming both C++ and Micropython. |
| Arduino | Yes(RTOS possible) | Yes (But with Jamaica no information is available) | Yes | No |
| STM32 | Yes(RTOS possible) | NA | [Yes(Possible)](https://micropython.org/download/) | No |

### 2.1.6 Selection of Real Time Application

Real Time applications are always time deterministic and user senses immediate responses from the system. The majorly well know real time applications are in Automotive sector (Cruise control system, Airbag system, ABS, etc), Space sector (Rovers, different electronics in the satellite, etc) all these system are called Real Time Application. Based on the applications a real time application can be divided into 3 different categories,

* Hard Real Time software systems have a set of strict deadlines, and missing a deadline is considered a system failure.
* Soft Real Time try to reach deadlines but do not fail if a deadline is missed.
* Firm real time systemstreatinformationdelivered/computationsmade after a deadline asinvalid.

Our goal was to decide upon a simple real time application which will have 2-3 threads running concurrently and need for synchronization. First there was no idea to implement the system using hardware components so, the decision was made to design for an Automotive Cruise Control system. But, later when hardware came into picture the components for Cruise control isn’t outsourceable. So, **Fire Detection and Alarm system** was picked up as a real time application since the components are easily gettable. Also this is a hard real time system and has more time deterministic value.

## 2.2 Literature Review

This section explicates the journals and technical papers referred briefly and it gives summary of all literature papers referred.

### 2.2.1 Overview of Papers and Journals Referred

**A large-scale study of programming languages and code quality in GitHub [Ray, B., Posnett, D., Devanbu, P. and Filkov, V., 2017]**

This paper focuses on large Scale Study of Programming Languages for Multithreaded programming and Code Quality from Github. Involving top 19 languages from Github in various language fields like procedural, scripting, object oriented etc. The language with the maximum number of source files is assigned as primary language for this project. Classification of different languages are performed, then were put into bugs analysis on different perspective like memory, concurrency, security and failure errors. Concurrency related was based on deadlock and race condition errors. In future language complexity gaps between different programming languages can be examined.

**About 15 years of Real-Time Java [Higuera-Toledano, T., 2012]**

This paper focuses about JAVA threads and POSIX threads. Different Java virtual machine for real time platform. This paper examines about use of JAVA threads for real time applications then conclusion is drawn where JAVA threads can't be used for RTA because of the memory management issues. So, POSIX threads were introduced which are native threads and that can also have connection with Kernel for RTA's. Discussed about existing flaws in non-real time java. Different Java virtual machine packages are discussed and their use in different fields are mentioned.  
**Event-Driven Multithreading Execution Platform for Real-Time On-Board Software Systems [Hammadeh, Z., Franz, T., Maibaum, O., Andreas, L., 2019]**This paper focuses on Optimizing an real time application (Optical navigation system) using event driven multithreaded programming. Multithreaded program is achieved using C++ language with use of POSIX threads. Event-driven multithreading execution platform and software development library: Tasking Framework is presented. It is dedicated to develop space applications which perform on-board data processing and sophisticated control algorithms, and have high computational demand. The future work is planned to make Tasking Framework open source and bare metal implementation.

**A Comparative Study of Programming Languages in Rosetta Code [Nanz, S. and Faria, C., 2015]**

In this paper, research is mainly focused on comparing different programming languages with various techniques. This study compares 8 widely used languages like (Python, C, C#, Go, Java,Pearl, Ruby, F#, Haskell) based on 7’087 solution programs corresponding to 745 tasks. All the different tasks are analyzed by Rosetta code.The experiments ran on a Ubuntu 12.04 LTS 64bit GNU/Linux box with Intel Quad Core2 CPU at 2.40 GHz and 4 GB of RAM. The programming languages are compared for,

* Language that makes most concise code. (Func and Script)
* Which language compiler into smaller executables.(bytecode)
* Better running time performance( C is king)
* Memory usage efficiency (Procedural lang).
* Less failure prone (Compile time errors more and GO wins)

Future work will be focused on pursuing best performance optimization in each language for multithreading programming.

**The role of concurrency in an evolutionary view of programming abstractions [Silvia, D., 2015]**

This paper aims at pointing out a number of remarks and connect concurrent programming languages under an evolutionary perspective, in order to grasp a unifying, but not simplistic, view of the programming languages development process. Majorly talks about evolution of programming after concurrency was brought in. Programming languages are split into old and new languages based on before concurrency and after concurrency. The languages are again split up into procedural, object oriented and scripting languages. The greatest cost of concurrency, that also limited its accessibility, is that (correct) concurrent programming is really hard and refactoring sequential code to add concurrency is even harder. Different models are discussed like,

* Shared memory- In this Java threads are not efficient they are error prone
* Message passing- Various languages like FORTRAN, Java, C and C# are discussed.
* GPU- Here CUDA programming language is discussed which is again based on C/C++.

Gaps in the study based on the role of concurrent program support for real time applications.

**Design of an empirical study for comparing the usability of concurrent programming languages [Nanz, S., Torshizi, F., Pedroni, M. and Meyer, B., 2013]**

This paper focuses on comparison between SCOOP and JAVA language for multithreading is discussed. A comparison is done using a multithreaded program (consumer and producer).

* Programmers can comprehend an existing program written in SCOOP more accurately compared to an existing program having the same functionality written in Java Threads (program comprehension). It is easier to program using SCOOP than using Java Threads.
* Programmers can find more errors in an existing program written in SCOOP than in an existing program of the same size written in Java Threads (program debugging).
* Programmers make fewer programming errors when writing programs in SCOOP than when writing programs having the same functionality in Java Threads (program correctness).

Future work focuses on determining if Java's behaviour is superior or indicative of concrete issues (compiler errors). Different concurrent multithreading language can be addressed by implementing a Real Time Application and comparing it. Gaps in the study based on the role of concurrent program support for real time applications

**COMPARATIVE ANALYSIS OF SOME PROGRAMMING LANGUAGES [Oyenike, B., 2012]**

This paper focuses on examining programming languages with different performance parameters. Examine the performance parameters like memory size and execution time for 6 languages namely C++, FORTRAN,PASCAL, BASIC, COBOL,JAVA.

* Object-Oriented program has the smallest running time followed by scientific program then Non-scientific.
* With the Object Oriented programming languages, JAVA program is faster than C++, although it required more memory allocation than all other programming languages.
* With the Scientific programming language, PASCAL is faster than FORTRAN but they both have the same memory space requirements.
* Lastly, in Non-scientific programming language, BASIC has lesser running time compare to COBOL and it also required less memory space allocation compared to COBOL.

The future work focuses on comparing different multithreading languages using different performance parameters.

**Python in Real Time Application for Mobile Robot [Mahmoud, A., Rabbah, N., Rabbah, B. and Mounir, R., 2018]**

This paper focuses on different python distributions like Cpython, micropython, jpython. Different problems in satisfying real time application are also discussed.

* Each actuator or sensor will be presented in COPDAI as Node. So, we will have just one process per task, and no need to use multithreading, against we will use multiprocessing, and so avoid the GIL downside, collaboration between these processes will not be discussed in this contribution.
* Different solution for Garbage collector and memory management are discussed.In this contribution they presented a major problem that face developers to write a real time application, and some best practices and design pattern to overcome this difficulty, some of them are general to any application, other are specific to Python programming language.
* A suitable architecture was presented to develop a mobile robot parts by using the Node pattern. This work will be reference for all developers that will contribute to COPDAI middleware development process.

The future work would be developing a core of the middleware, performing tests on mobile robots, and to build a large community around this project.

**Realtime Applications with RTMaps and Bluebox 2.0 [Sreeram, V., Surya, M., Akash, G., Niranjan, R., Sree, B., Shruthi, B. and Mohamed, S., 2018]**

This paper focuses on using Python in Real-Time maps. Methodologies used is using Neural network and taking a lok at what kind of Image it is.

* Python component of RTMaps embedded packages is used to develop a neural network and do the classification of the input image.
* The python block of RTMaps make a class called RTMaps\_python. This can be made to call reactively to an input or periodically.
* The core function inside the RTMaps\_python class act as infinite loop and this is where main program is running. The vehicle detection and Traffic sign classification are the two examples implemented in python.
* The second project with RTMaps-Embedded python, is for traffic sign (Germany) classification using tensor flow model. This is made to identify the traffic sign board, classify it in to one of the 43 classes and display the meaning of detected traffic sign board from the image. The classification model used ~39000 images to train the model. The tensor flow model is recreated for testing and the model weights, saved using the training model, are added to predict the class of input image. This entire test structure is done in the RTMaps python component.

**Automatic Image Quality Assessment in Digital Pathology: From Idea to Implementation [David, A., Christophe, D., Valerie, P., Fatiha, B., Maxime, B., Luc, Legres., Anne, J., Philippe, B. and Jean, Y., 2014]**

This paper focuses on creating libraries to assess the automatic image detection using Python and Java.

As quality assurance is crucial in a context of daily use in diagnostic pathology, here a fast and reliable no-reference quality assessment library for WSI has been developed and digital images in general. Development of Service Providing Interfaces and Application Programming Interfaces have been carried out in 2012-2014, and implementation started in French national projects in 2013. Applications based on these libraries can be used upstream, as calibration and quality control tool for the WSI acquisition systems, or as tools to reacquire tiles while the WSI is being scanned.

Such quality assessment scores could be integrated as WSI’s metadata shared in clinical, research or teaching contexts, for a more efficient medical informatics workflow which can be a future work.

**Real-Time Operating System FreeRTOS Application for Fire Alarm Project in Reduced Scale. [de Oliveira Turci, Luca. (2017)]**

An RTOS is used for implementing Fire Alarm project. Different sensors are used to sense the fire and smoke.

* Different sensors are used to detect smoke and flam like MQ-2 and YS17. The sensors and actuators are given as FreeRTOS tasks.
* In the present project, the performance of the FreeRTOS kernel using Arduino Nano board was analyzed together with external equipments (oscilloscope and function generator) and some practical methods through several parameters.
* Considering the circumstances of the experiments, it is possible to conclude that the FreeRTOS kernel really presented determinism and reliability. Jitter was estimated around 25 us considering within the criteria that defines a hard real-time system as a system which has a jitter no higher than 100μs in tasks that has cycles of up to 10ms.

**Development of Fire Alarm System using Raspberry Pi and Arduino Uno [Saifudaullah, B., Rosni, K., 2013]**

The techniques of Image detection and alerting using SMS is used. The proposed Fire alarm system is a real-time monitoring system that detects the presence of smoke in the air due to fire and captures images via a camera installed inside a room when a fire occurs. The embedded systems used to develop this fire alarm system are Raspberry Pi and Arduino Uno. The key feature of the system is the ability to remotely send an alert when a fire is detected. When the presence of smoke is detected, the system will display an image of the room state in a webpage and send an SMS Firefighter. The advantage of using this system is it will reduce the possibility of false alert reported to the Firefighter. The camera will only capture an image, so this system will consume a little storage and power. The developed prototype offered a feature that enabled verification that a fire actually occurred. The fire alarm system warns the user by first sending an alert and asks for confirmation before submitting a report/alert to the “Fire-fighter”.

**An IoT based Fire Detection, Precaution & Monitoring System using Raspberry Pi3 & GSM [Kulkarni, S., Prasanna, T., and Bramaramba, K., 2019]**

Image detection technique is used for fire detection. In this paper, the propounded system is capable to detect fire and can provide the location of the affected region. Raspberry Pi 3 has been used to control multiple Node MCU which are integrated with a couple of sensors. A 360° relay motor is assembled with the camera so that it can snap the image in whatever angle the fire is detected. The sensor data values & images always update on webpage. The confirmation of the fire suspecting system to avoid any false alarm. The system will immediately send a message along with the image of the affected spot and location. Further extend is by adding some more features which can make it more efficient and security oriented. The camcorder can also be used to track all the activities of the unknown person or intruders.

### 2.2.2 Literature Summary

In this section the summary of all the journals and papers are represented in the form of table which is shown in Table 2.3.

Table 2. 4: Literature summary for referred papers

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Authors** | **Journal name & Year of Publication** | **Research Focus** | **Methods and Methodo-logies used** | **Research Findings** | **Conclusions drawn by authors** | **Limitations of Study** | **Critical Appraisal of the Published Work ( by the student)** |
| 1 | Baishakhi Ray, Daryl Posnett, Vladimir Filkov, Premkumar Devanbu | Communications of the ACM, 60(10), pp.91-100, 2017 | Focused on large Scale Study of Programming Languages for Multithreaded programming and Code Quality in Github | Involving top 19 languages from Github in various language fields like procedural, scripting, object oriented etc. The language with the maximum number of source files is assigned as primary language for this project. | Classification of different languages are performed, then were put into bugsanalysis on different perspective like memory, concurrency, security and failure errors. Concurrency related was based on deadlock and race condition errors. | Statically typed languages in general are less  defect prone than the dynamic types, and that strong typing is better  than weak typing in the same regard. External tools also impact software  quality; Go has lot more concurrency bugs related to race condition  due to its race condition detection tool. Software components  developed in C++ are in general more reliable than C. | Since the datasets from Github are being sliced and diced it is unable to quantify the specific effects of language type on usage. | Could’ve be tested with some hardware components. |
| 2 | M. Teresa Higuera-Toledano | Madrid 28040, Spain: Universidad Complutense de Madrid. | Focused on JAVA threads and POSIX threads.Different Java virtual machine for real time platform | This is a survey paper no methods are used here. | This paper examines about use of JAVA threads for real time applications. So, POSIX threads were introduced which are native threads and that can also have connection with Kernel for RTA's. Discussed about existing flaws in non-real time java. | Conclusion is drawn where JAVA threads can't be used for RTA because of the memory management issues. Different Java virtual machine packages are discussed and their use in different fields are mentioned. | - | Testing with RTA would’ve brought in better understanding |
| 3 | Hammadeh, Zain & Franz, Tobias & Maibaum, Olaf & Gerndt, Andreas & Lüdtke, Daniel | - | Focused on optimizing an real time application(Optical navigation system) using event driven multithreaded programming. | Multithreaded program is achieved using C++ language with use of POSIX threads. | It is dedicated to develop space applications which perform on-board data processing and sophisticated control algorithms, and have high computational demand. | Event-driven multithreading execution platform and software development library: Tasking Framework is presented. | The framework could have been explained in a better way and the slight hint about the logic could have been given. | More functionalities can be added with the developed system |
| 4 | Sebastian Nanz, Carlo A. Furia | 2015 IEEE/ACM 37th IEEE International Conference on Software, [online] pp.778-788 | Focused on Comparing different programming languages with various techniques | This study compares 8 widely used languages like (Python, C, C#, Go, Java,Pearl, Ruby, F#, Haskell) based on 7087 solution programs corresponding to 745 tasks. The experiments ran on Ubuntu 12.04 LTS 64bit GNU/Linux box with Intel Quad Core2 CPU at 2.40 GHz and 4 GB of RAM. | The programming languages are compared for,  Language that makes most concise code. Which language compiler into smaller executables. Better running time performance. Memory usage efficiency. Less failure prone. | Functional and scripting language makes more concise code. Bytecode languages like Python or Java makes more smaller executables. C programming is the king in better running time performance. Procedural language has more memory efficiency. GO language has less compile time errors comparing all. | - | Scalability of the system can be improved by implementing new algorithms for gateway medium |
| 5 | Silvia Crafa | arXiv:1507.07719v1 [cs.PL] 28 Jul 2015 | It aims at pointing out a number of remarks and connect concurrent programming languages under an evolutionary perspective, in order to grasp a unifying, but not simplistic, view of the programming languages development process. | Programming languages are split into modern and new languages based on before concurrency and after concurrency. The languages are again split up into procedural, object oriented and scripting languages. | Majorly talks about evolution of programming after concurrency was brought in. The greatest cost of concurrency, that also limited its accessibility, is that (correct) concurrent programming is really hard and refactoring sequential code to add concurrency is even harder. | Different models are discussed like,  Shared memory- In this Java threads are not efficient they are error prone.  Message passing- Various languages like FORTRAN, Java, C and C# are discussed.  GPU- Here CUDA programming language is discussed which is again based on C/C++. | Different programming languages could have been explained with an example which will bring a better understanding. | Can add some performance analysis tools |
| 6 | Sebastian Nanz, Faraz Torshizi, Michela Pedroni, Bertrand Meyer | Information and SoftwareTechnology, 55, pp.1304-1315. | Focuses on comparison between SCOOP and JAVA language for multithreading is discussed. | A comparison is done using a multithreaded program (consumer and producer) | Programmers can comprehend an existing programwritten in SCOOP more accurately compared to an existing program having the same functionality written in Java Threads (program comprehension). It is easier to program using SCOOP than using Java Threads. | Programmers can find more errors in an existing program written in SCOOP than in an existing program of the same size written in Java Threads (program debugging).Programmers make fewer programming errors when writing programs in SCOOP than when writing programs having the same functionality in Java Threads (program correctness). | More errors related to concurrency could have been discussed. | Scalability and accuracy of the system can be improve by implementing new algorithms for gateway medium |
| 7 | Oguntunde, Bosede Oyenike | Transnational Journal of Science and Technology, [online] 2(5), pp.107-118. | Examine programming languages with different performance parameters | Examine the performance parameters like memory size and execution time for 6 languages namely C++, FORTRAN,PASCAL, BASIC, COBOL,JAVA. | Object-Oriented program has the smallest running time followed by scientific program then Non-scientific.With the Object Oriented programming languages, JAVA program is faster than C++, although it required more memory allocation than all other programming languages. | With the Scientific programming language, PASCAL is faster than FORTRAN but they both have the same memory space requirements. Lastly, in Non-scientific programming language, BASIC has lesser running time compare to COBOL and it also required less memory space allocation compared to COBOL. | - | - |
| 8 | Mahmoud, Rabbah, Nabila, Rabbah, Hicham, Belhadaoui, Mounir, Rifi | Smart Application and Data Analysis for Smart Cities (SADASC'18). | Focused on python In Real Time Application For Mobile Robot | Different python distributions like Cpython, micropython, jpython is discussed. Different problems in satisfying real time application are mentioned. | In this contribution they presented a major problem that face developers to write a real time application, and some best practices and design pattern to overcome this difficulty, some of them are general to any application, other are specific to Python programming language. | Different solution for Garbage collector and memory management are discussed. | The major problems are addressed here but the solution for the problem isn’t discussed. | The different types of python could be examined with software development |
| 9 | Sreeram Venkitachalam, Surya Kollazhi Manghat, Akash Sunil Gaikwad, Niranjan Ravi, Sree BalaShruthi Bhamidi and Mohamed El-Sharkawy | International Conference of Artificial Intelligence 2018. | Focuses on using Python in Real-Time maps | Methodologies used are using Neural network for taking a look at what kind of Image it is. | Python component of RTMaps embedded packages is used to develop a neural network and do the classification of the input image.The python block of RTMaps make a class called RTMaps\_python. This can be made to call reactively to an input or periodically. | The core function inside the RTMaps\_python class act as infinite loop and this is where main program is running. The vehicle detection and Traffic sign classification are the two examples implemented in python. | The role about RTmaps in Autonomous cars aren’t discussed. | - |
| 10 | David Ameisen, Christophe Deroulers, Valérie Perrier, Fatiha Bouhidel, Maxime Battistella, Luc Legrès, Anne Janin, Philippe Bertheau, Jean-Baptiste Yunès. | Sorbonne Paris Cité, 75205 Paris Cedex 13, France | Focuses on automatic Image Quality Assessment in DigitalPathology: From Idea to Implementation | Libraries are being created to assess the automatic image detection using Python and Java | As quality assurance is crucial in a context of daily use in diagnostic pathology, here a fast and reliable no-reference quality assessment library for WSI has been developed and digital images in general. Applications based on these libraries can be used upstream, as calibration and quality control tool for the WSI acquisition systems, or as tools to reacquire tiles while the WSI is being scanned. | The developed library ensures that there is automatic image quality assessment. | **-** | - |
| 11 | Kulkarni Sangam, T. Prasanna, K. Bramaramba | INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 08, Issue 07 (July 2019) | Focused on an IoT based Fire Detection, Precaution & Monitoring System using Raspberry Pi3 & GSM | Image detection technique is used for fire detection. | In this paper, the propounded system is capable to detect fire and can provide the location of the affected region. Raspberry Pi 3 has been used to control multiple Node MCU which are integrated with a couple of sensors. A 360° relay motor is assembled with the camera so that it can snap the image in whatever angle the fire is detected. | The sensor data values & images always update on webpage. The confirmation of the fire suspecting system to avoid any false alarm. The system will immediately send a message along with the image of the affected spot and location. | What is the response time of capturing the fire isn’t mentioned. | The system could have been done using RTOS |
| 12 | Luca de Oliveira Turci | International Journal of Computing and Digital Systemss. 6. 198-204. 10.12785/IJCDS/060405 | Real-Time Operating System FreeRTOS Application for Fire Alarm Project in Reduced Scale | An RTOS is used for implementing Fire Alarm project. Different sensors are used to sense the fire and smoke. | Different sensors are used to detect smoke and flame. The sensors and actuators are given as FreeRTOS tasks. In the present project, the performance of the FreeRTOS kernel using Arduino Nano board was analyzed together with external equipments (oscilloscope and function generator) and some practical methods through several parameters. | Considering the circumstances of the experiments, it is possible to conclude that the FreeRTOS kernel really presented determinism and reliability. Jitter was estaminated around 25us considering within the criteria that defines a hard real-time system as a system which has a jitter no higher than 100μs in tasks that has cycles of up to 10ms. | - | Could have used temperature sensor to make the 100% presence confirmation of the Fire. |
| 13 | Saifudaullah Bin Bahrudin, Rosni Abu Kassim |  | Development of Fire Alarm System using Raspberry Pi and Arduino Uno | The techniques of Image detection and alerting using SMS is used. | The proposed Fire alarm system is a real-time monitoring system that detects the presence of smoke in the air due to fire and captures images via a camera installed inside a room when a fire occurs. The embedded systems used to develop this fire alarm system are Raspberry Pi and Arduino Uno. The key feature of the system is the ability to remotely send an alert when a fire is detected. | When the presence of smoke is detected, the system will display an image of the room state in a webpage and send an SMS FireFighter. The advantage of using this system is it will reduce the possibility of false alert reported to the Firefighter. | The presence of fire can actually burn the camera as well. Not sure how can a camera stay for a long time when there is fire. | The system could have been done using RTOS |

## 2.3 Problem Formulation

### 2.3.1 Identification of Research Gaps

* From the literature survey, it is evident that the developed systems (Fire detection and Alarming system) just focus on machine learning with camera based techniques.
* Hence, the proposed dissertation work emphasis on the Fire detection and Alarming system that will be developed in both C++ and Micropython by adding some real time capabilities to the chosen application. The developed code will be tested for Jitter, execution time, Memory consumption, Latency and WCRT.

### 2.3.2 Research questions the dissertation would like to address

* Compatibility of Hardware(microcontroller) for different languages.
* How to use FreeRTOS for developing on both Micropython and C++.
* Usage of ESP32 microcontroller for developing both Micropython and C++.
* Process to design a real-time application using Micropython.
* Ways to perform the Multithreading in Micropython.

# 3. Problem Statement

The chapter comprises of the title, aim, objectives and methodology of the Dissertation. The objectives of the project and the methods followed to achieve the identified objectives are listed.

## 3.1 Title

Performance Analysis of a concurrent Fire detection and Alarming system.

## 3.2 Aim

To compare the performance of Concurrent Fire Detection and Alarming system by providing a real time platform for both Python and C++ languages using FreeRTOS.

## 3.3 Project Objectives

Based on literature review, problem statement and aim of the project, objectives are identified and listed as below,

* To conduct a literature survey on POSIX threads and Python threads for its adaptablity with real time properties and its suitability of real time platform
* To arrive at the hardware and software requirements and design specification for a multithreaded Fire detection and Alarm system
* To design a real time application for fire detection and alarm system
* To implement sequential Fire detection and Alarm system using C++ and Python programming languages separately on the selected real time platform
* To implement multithreaded Fire detection and Alarm system program using C++ POSIX threads and Python threads separately on the selected real time platform
* To Compare the performance of Fire detection and Alarm system for the above implemented applications
* To document the report by unifying all the results and outcomes

## 3.4 Methods and Methodology

Table 3. 1: Methods and Methodology to achieve defined objectives

|  |  |  |
| --- | --- | --- |
| **Objective No.** | **Statement of the Objective** | **Method/ Methodology** |
| 1 | To conduct a literature survey on POSIX threads and Python threads for its adaptablity with real time properties and its suitability of real time platform | Literature review of previous work using books, journals and videos |
| 2 | To arrive at the hardware and software requirements and design specification for a multithreaded Fire detection and Alarm system | Based on the application and literature review the hardware and software requirements are concluded. |
| 3 | To design a real time application for fire detection and alarm system | Based on the requirements and specifications the algorithm and hardware interface is documented. |
| 4 | To implement sequential Fire detection and Alarm system using C++ and Python programming languages separately on the selected real time platform | C++ coding is done in Arduino IDE and Micropython in Visual Studio Code. |
| 5 | To implement multithreaded Fire detection and Alarm system program using C++ POSIX threads and Python threads separately on the selected real time platform | FreeRTOS is used to implement both C++ and Micropython. Additionally for micropython import\_thread library is used to add preemptive scheduling. |
| 6 | To Compare the performance of Fire detection and Alarm system for the above implemented applications | Compared for parameters like execution time, Jitter, Latency, memory allocated, WCRT. Results for existing and proposed system are compared for Jitter, WCRT and Latency |
| 7 | To document the report by unifying all the results and outcomes | All the results, design and implementations are documented. |

# 4. Design and Implementation

This chapter encompasses requirements analysis, design specifications, hardware and software requirements. Also it contains design stages block diagram, flow chart, algorithms and implementation.

## 4.1 Requirement analysis

This section contains some details about functional and nonfunctional requirements of the system designed.

### 4.1.1 Functional Requirements

Fire detection and Alarm system can be designed after doing analysis of basic functional requirements for system, it can be given as listed below,

1. The system should provide higher preference to Flame sensor to check whether fire has occurred or not.
2. The system should keep an eye on the water level in the tank all the time and notify when the water level is low.
3. When the manual press is attempted irrespective of any sensor values the system should actuate the water opening, turn ON the cooler fan, shut down the power and a louder alarm.
4. If the system gets an input from flame sensor or other sensors it should actuate the following based on the priority given.
5. System has to be tested for parameters like execution time, memory consumption, Jitter, Latency and WCRT other test cases like code coverage, correctness are to be analysed.

### 4.1.2 Non-functional Requirements

1. The system should be compact and not occupy much space.
2. The system should be time deterministic and there must not be any lag in water opening or flame sensing.
3. The system should always be in ON state.

## 4.2 System Specifications

A fire detection and alarming system can be very useful, since it helps in detecting the fire in each and every building. Though this system is designed with time deterministic nature in mind, it acts as a life saver every now and then.

### 4.2.1 Hardware requirements

Hardware requirements defines actual idea of how system will be more efficient and effective. As per the desired outcomes and required features in the system, hardware requirements are necessary to decide. In this case of Fire detection and Alarming system sensors and plays a major role. Based on the sensor values the actuator plays around. Based on the sensors and processors comparisons, this section gives better idea of choosing an appropriate hardware. Some of the available hardware for system developed can be explained.

#### 4.2.1.1 Embedded boards:

In market today so many system on chip devices available with different functionalities. Some of these hardware can be given as, Beagle bone, Raspberry Pi, Arduino, Microcontrollers etc. based on the system requirement to choose an appropriate hardware let’s consider comparison stages based on its features and specifications like processor speed, supported operating system and languages, memory space, USB etc. Table 4.1 shows the comparison of all these boards with its specifications.

Table 4. 1: Embedded boards comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Platform** | **Arduino** | **ESP32** | **Beagle Board** | **Raspberry pi** | **STM32** |
| Variant | Uno | Dev module | Beagle Bone black | Model-3B+ | F446RE Nucleo |
| **Software** | | | | |  |
| Operating System | - | - | Android, Linux, Windows CE, RISC Os | Linux, RISC OS | - |
| Toolkit | Arduino IDE, Eclipse | Arduino IDE/ESP IDF, Visual Studio Code. | Eclipse, Android ADk, Scratchbox | OpenEmbedded, QEMU, Scratchbox, Eclipse | STMCube MX IDE/Arduino IDE |
| Programming language | Wiring based Embedded C++ | Embedded C++, Python, Embedded C. | Python, Embedded C, etc. | Python, c, Possibly BASIC | Embedded C, Python. Embedded C++ |
| Architecture | 8 Bit | 32 Bit | 32 Bit | 64 Bit | 32 bit |
| **Hardware** | | | | |  |
| Processor | ATMEGA 328 | Extensa LX6 processor | ARM cortex A8 | ARM cortex A53 | ARM cortex M4 processor |
| Speed | 16Mhz | 80-240Mhz | 1GHZ | 1.4Ghz | 84Mhz |
| RAM | 2 kB | 520kB | 256MB | 1GB | 96KB |
| ROM | 32 kB | 4MB/8MB/16MB | 256MB Flash | SD | 512KB |
| I/O(various protocol) | 14 pins | 36 pins |  | 40 pins | 50 pins |
| ADC | 6 bit | Internally used | Internally used | Internally used | 12 bit |
| USB | - | - | 1 × 2.0 | 2 × 2.0 | 1 x 2.0 |
| Audio | - | - | Stereo In/Out | Stereo Out, In/USB Mic | - |
| Video | - | CAM video streaming | DVI-D, S-Video | HDMI,NTSC or PAL | CAM video streaming |
| Misc. | Many shields available for capability | Many shields available for capability | SD/MMC, RS-232, JTAG, USB, OTG, LCD | SD, 10/100 Ethernet, JTAG | Many shields available for capability |
| COST | Rs. 400 | Rs.500 | Rs.6500 | Rs.2575 | Rs.1500 |

* **ESP32 Dev module board:**

ESP32 is a powerful microcontroller with 4/8/16MB flash memory. Since there are many options for flash memory it is very useful for different Embedded and IoT related projects. Xtensa LX6 is a powerful processor with inbuilt Wifi and BLE modules within the core itself. There are about 36 pins which makes it unique since it’s compact and has almost all the features. There are different communication protocols like I2C, SPI, CAN, UART. Different programming languages that can be used to program ESP32 like C, C++, Python Java.

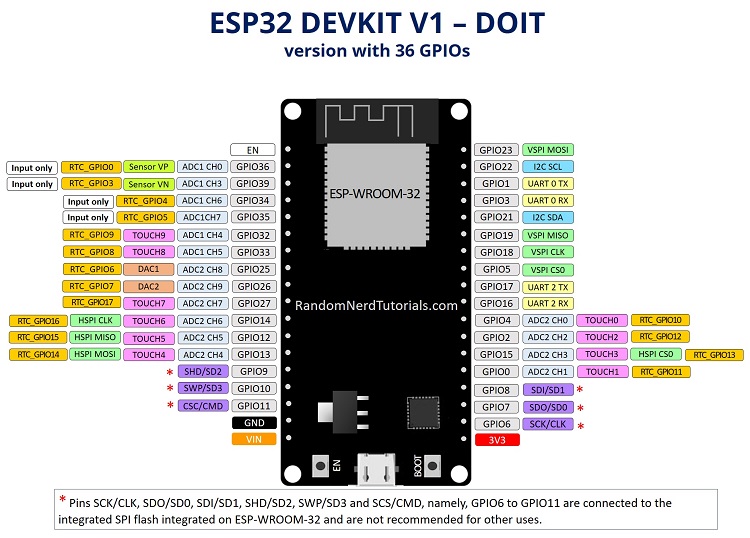


Figure 4. 1: ESP32 Dev module

#### 4.2.1.2 Sensors:

Sensors are the ones that acts as an input to sense or measure a given threshold agreed by the programmer and communicates to the microcontroller that this has went beyond the control. Microcontroller takes that input and does the proceedings. The sensors are ruling this work to make the things smarter. Right from normal smartphones to minute parts inside engine everywhere sensors are present.

**Flame sensor:**

A flame sensor module that consists of a flame sensor (IR receiver), resistor, capacitor, potentiometer, and comparator LM393 in an integrated circuit. It can detect infrared light with a wavelength ranging from 700nm to 1000nm.The far-infrared flame probe converts the light detected in the form of infrared light into current changes. Sensitivity is adjusted through the onboard variable resistor with a detection angle of 60 degrees.

Working voltage is between 3.3v and 5.2v DC, with a digital output to indicate the presence of a signal. Sensing is conditioned by an LM393 comparator.

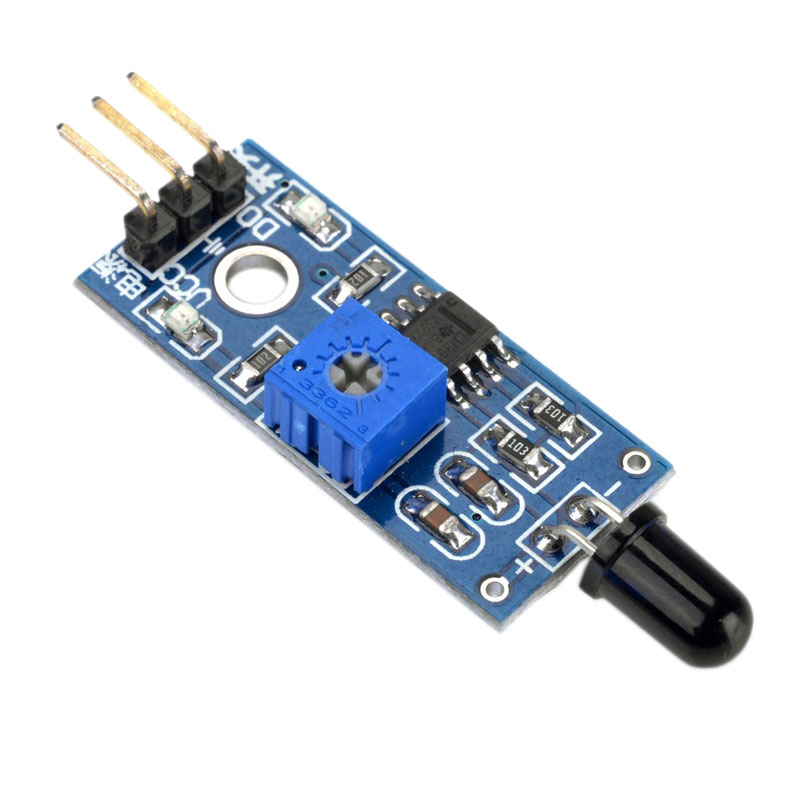


Figure 4. 2: Flame sensor YS-17

**Water level sensor:**

The Water Level Depth Detection Sensor for Arduino has Operating voltage DC3-5V and Operating current less than 20mA. The Sensor is the Analog type which produces analog output signals according to the water pressure with its Detection Area of 40x16mm.

**The Water Level Sensor is an easy-to-use and cost-effective with high level/drop recognition sensor by having a series of parallel wires exposed traces measure droplets/water volume in order to determine the water level.**

Easy to complete water to analog signal conversion and output analog values can be directly read Arduino development board to achieve the level alarm effect.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

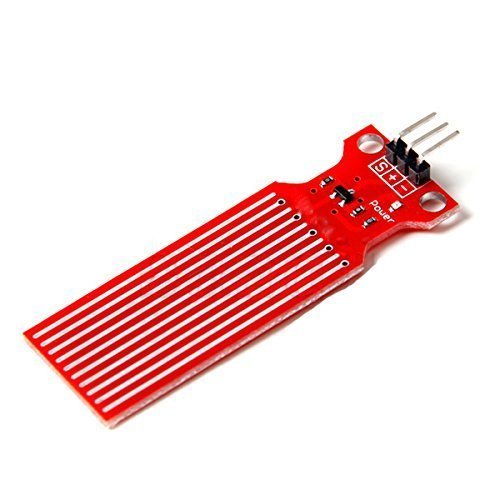


Figure 4. 3: Water level sensor

**MQ-2 Gas sensor:**

The MQ-2 Smoke LPG Butane Hydrogen Gas Sensor Detector Module is useful for gas leakage detection (home and industry). It is suitable for detecting H2, LPG, CH4, CO, Alcohol, Smoke or Propane. Due to its high sensitivity and fast response time, measurement can be taken as soon as possible. The sensitivity of the sensor can be adjusted by the potentiometer.

MQ-2 gas sensor using gas sensitive material is to be clean air in the lower conductivity of Tin oxide (SnO2). When the sensor when flammable gases are present in the environment in which the conductivity of the sensor with an increasing concentration of combustible gas in the air increases.

Use a simple circuit to convert the changes in conductivity and output signal that corresponds to the concentration of the gas.

MQ-2 gas sensor higher sensitivity to liquefied petroleum gas, propane, hydrogen, detection of gas and other combustible vapors are ideal. This sensor can detect a variety of flammable gas, is a low-cost sensor for many applications.



Figure 4. 4: Smoke sensor MQ-2

#### 4.2.1.3 Actuators:

An actuator is a component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. An actuator is the mechanism by which a control system acts upon an environment. The control system can be simple (a fixed mechanical or electronic system), software-based (e.g. a printer driver, robot control system), a human, or any other input.

**Motor (for exhaust fan):**

A DC motor is an electric motor which converts electric energy into mechanical energy. The physical principle of all electric motors is that when an electric current is passed through a conductor (usually a coil of wire) placed within a magnetic field, a force is exerted on the wire causing it to move. The classic DC motor has a rotating armature in the form of an electromagnet. A rotary switch called a commutator reverses the direction of the electric current twice every cycle, to flow through the armature so that the poles of the electromagnet push and pull against the permanent magnets on the outside of the motor. As the poles of the armature electromagnet pass the poles of the permanent magnets, the commutator reverses the polarity of the armature electromagnet. During that instant of switching polarity, inertia keeps the motor going in the proper direction. The DC motor uses 12V for its operation. In our project DC motor is used to show the movement of the car.

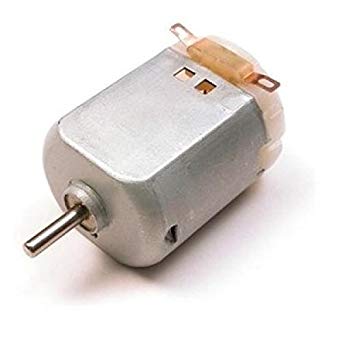


Figure 4. 5: DC motor

**LED:**

Light Emitting Diode which is commonly known as LED is semiconductor device that emits light when an electric current is passed through it. In Simple terms it is defined as a Semiconductor device that converts light energy into electrical energy. Light is produced when the holes and electrons combine together within solid semiconductor material and hence these are also called as Solid state devices. Light Emitting Diodes are made from a very minute layer of heavily doped semiconductor material and depending on the semiconductor material and the amount of doping, when forward biased the LED will emit a light based on different spectral wavelength. Most common LED’s require operating voltage between 1.2V to 3.6 volts.

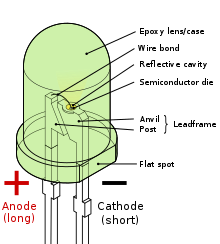


Figure 4. 6: Light Emitting Diode

**Buzzer:**

A **buzzer** or **beeper** is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (*piezo* for short). Piezoelectric buzzers, or piezo buzzers, as they are sometimes called, were invented by Japanese manufacturers and fitted into a wide array of products during the 1970s to 1980s. This advancement mainly came about because of cooperative efforts by Japanese manufacturing companies. In 1951, they established the Barium Titanate Application Research Committee, which allowed the companies to be "competitively cooperative" and bring about several piezoelectric innovations and inventions



Figure 4. 7: Buzzer

**Submersible water pump:**

A submersible pump is a device which has a hermetically sealed motor close-coupled to the pump body. Submersible pump cable are designed for use in wet ground or under water. In this project the water pump is used to show realistic view of the water opening when a fire occurs. Operating voltage will be 3-6V.



Figure 4. 8: Submersible water pump

#### 4.2.1.4 Laptop:

To simulate the system and for the result analysis to be presented with all functionality designed minimum requirements for laptop are,

* Windows 10 or above version
* 64-Bit operating system
* RAM: 4 Giga Bytes

### 4.2.2 Software requirements

#### 4.2.2.1 Arduino IDE:

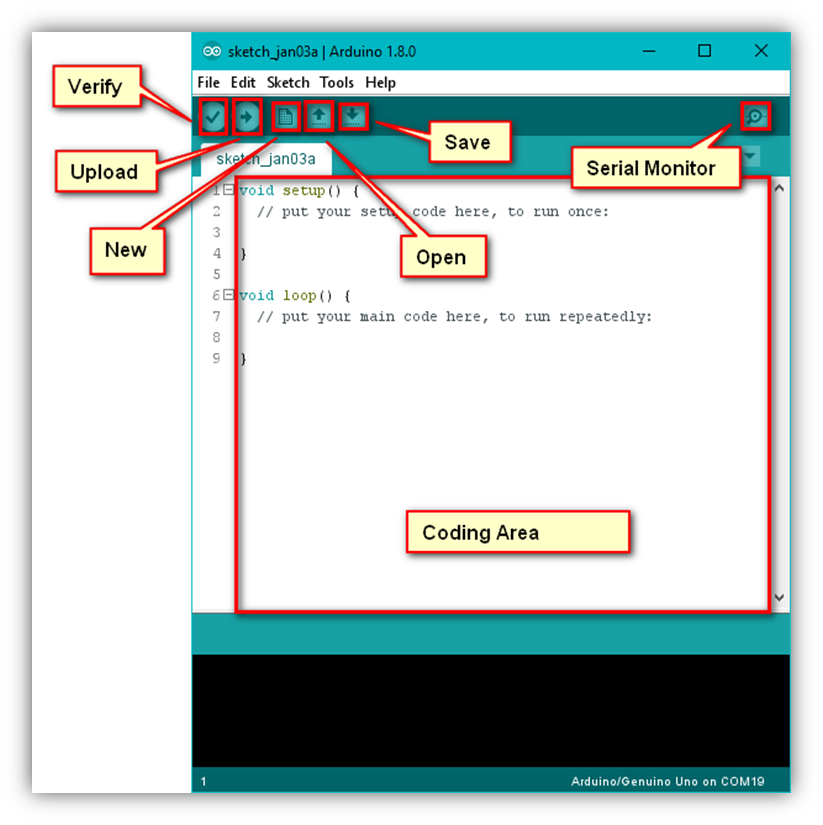
Arduino IDE is an open-source electronics platform, it is growing through the contribution of user worldwide based on easy-to-use hardware and software. Arduino IDE script written in JAVA based on processing programming, AVR-gcc, and other open source platform and it is translated into C language. Programmer can do programming in Arduino programming language and according to wiring, and Arduino software is based on processing programming environment. Arduino (IDE) runs on Mac, windows and Linux operating system. User can easily add AVR C code with Arduino program. ****

Figure 4. 9: Arduino IDE UI

#### 4.2.2.2 Visual Studio Code:

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and MacOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. Visual Studio Code's source code comes from Microsoft's free and open-source software **VSCode** project released under the permissive Expat License, but the compiled binaries are freeware for any use.

### 4.2.3 Selected System components and Development costing

From the given functionalities of system and based on all the requirements, chosen hardware and software listed. As shown, ESP32 board have been chosen as hardware and as it is cost effective and compatible with both C++ as well as Micropython compared to other boards.

Table 4. 2: Hardware and software costs

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **System Components** | **Functionality** | **Cost** |
| 1. | Arduino IDE | Used for C++ Programming | Student edition |
| 2. | Visual Studio Code | Used for Micropython programming | Student edition |
| 3. | ESP32 Dev module | Used for developing Fire detection and monitoring system | 500/- |
| 4. | Flame Sensor (IR based) | To sense the fire | 230/- |
| 5. | Water level sensor (ROBODO - SEN18) | To Measure the water level | 180/- |
| 7. | [MQ-2 Gas Sensor Module](http://www.icstation.com/product_info.php?products_id=3025) | To sense the smoke | 210/- |
| 8. | Water Pump | To perform water opening | 70/- |
| 9. | DC motor(as exhaust fan) | Used for letting out smoke | 50/- |
| 10. | Push button | For manual press | 20/- |
| 11. | LED’s(for indication) | For indication | 20/- |
| 12. | Buzzer | For Alarming | 30/- |
| 13. | Connecting wires | To connect all sensors with ESP32 | 50/- |
| TOTAL COST FOR DEVELOPMENT OF SYSTEM | | | 1360/- |

## 4.3 Design Specification

1. When the system is initialized, the sensor senses for fire/smoke and water level in the environment and reports to the microcontroller.
2. Fire sensor just senses for the fire if it finds fire its 1 detected and if there is no fire its 0. The fire sensor and water level sensor would be given highest priority followed by smoke sensor.
3. Calculated sensor values are given to the ESP32 every now and then and controller assess the values.
4. If the threshold for the sensors reaches, immediately controller sends signal to the actuator for water opening, turn ON exhaust fan and Buzzer with LED indication.
5. If the manual button press is attempted irrespective of sensor values all the actuations has to take place like water opening, exhaust fan, light alert and Alarming.
6. The same design will used in 4 different ways sequential C++, multithreading C++, sequential Micropython and multithreading with Micropython.

## 4.4 System Design

System design gives exact idea for the implementation and development stages. It includes, Block diagram, data flow chart, algorithms etc.

### 4.4.1 Block diagram of system

The system can be identified with the flow and directions of its connections with other peripherals and sub components in the system. For the developed system, figure 1 shows the high level block diagram of the Fire detection and Alarming system with inputs provided to the controller unit for desired results at end of the system. These inputs are nothing but different sensors like flame sensor, water level measuring sensor, smoke detection sensor and temperature sensor. All the sensor values are provided to the microcontroller and microcontroller processes it if, at all the values are more than the threshold then needed actions are taken which are present in the output actuation side

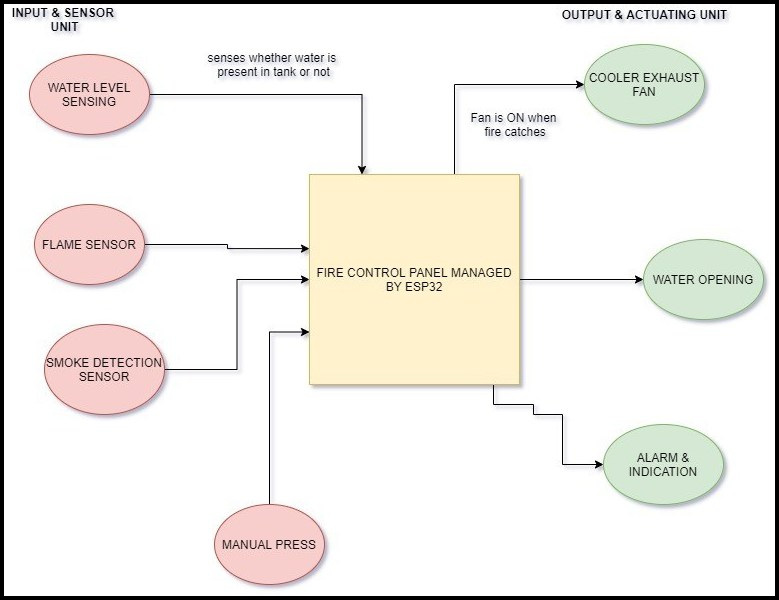


Figure 4. 10: High level Block diagram

The figure 2 represents the low level block diagram for Fire detection and Alarming system. It depicts like what work each sensor performs right from sensing water, detecting fire, detecting smoke, sensing the temperature reading. There is a manual press button, if that is pressed irrespective of any sensor values the output actuation unit will start based on the priority given.

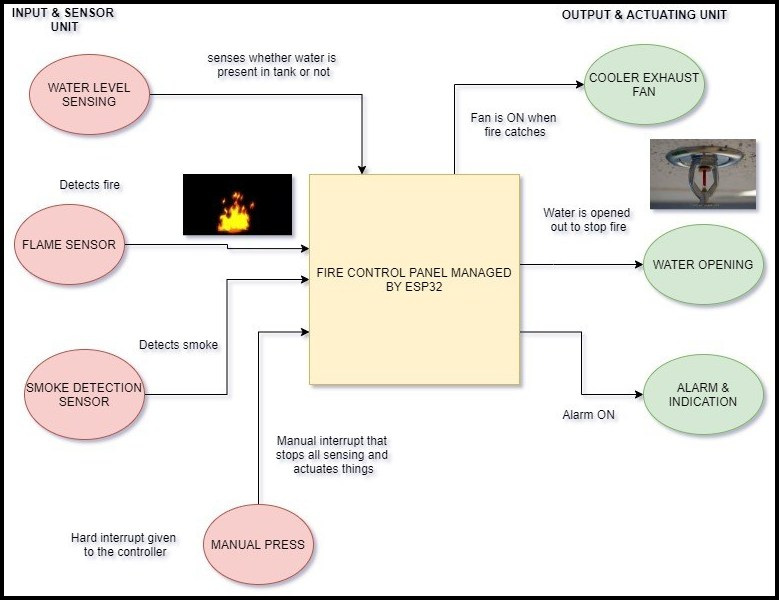


Figure 4. 11: Low Level Block Diagram

### 4.4.2 Schematic Diagram

Schematic diagram provides visual presentation of the system that describe the interconnection between each sub system and it has enough detail to connect an electronic circuit.

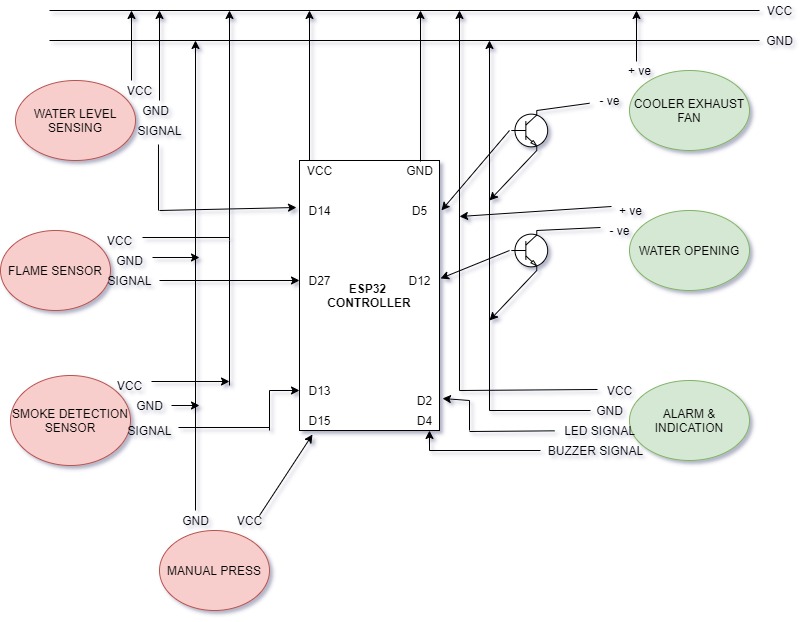


Figure 4. 12: Schematic Diagram

In Figure 4.12 schematic diagram of the system, smoke sensor and flame sensor fetches the data from the environment and feeds to the microcontroller based on that the actuation takes place(Water opening, Exhaust fan and indication). Irrespective of anything the water level sensor senses the water level in the tank and provides the reading based on which it is considered to be high, low or medium. The manual press button is just a switch when it is turned ON irrespective of sensor values the system actuates.

### 4.4.3 Flow chart of the system

The flowchart explains the whole working of the Fire detection and Alarm system. When the system is initiated it checks for all the sensor values. If at all the sensor values are abnormal (above the threshold limit) then it actuates the water opening, exhaust fan and notifies by Alarming. There is another part for this irrespective of sensor values if the manual press button is attempted it actuates the water opening, exhaust fan and notifies by Alarming.



Figure 4. 13: Flow chart

## 4.5 Implementation of System

The implementation is divided into two parts hardware and software implementation. The software implementation is done using Arduino IDE (C++), Visual Studio Code (Micropython). And hardware implementation includes the sensor, actuators and Microcontroller board.

### 4.5.1 Hardware Setup:

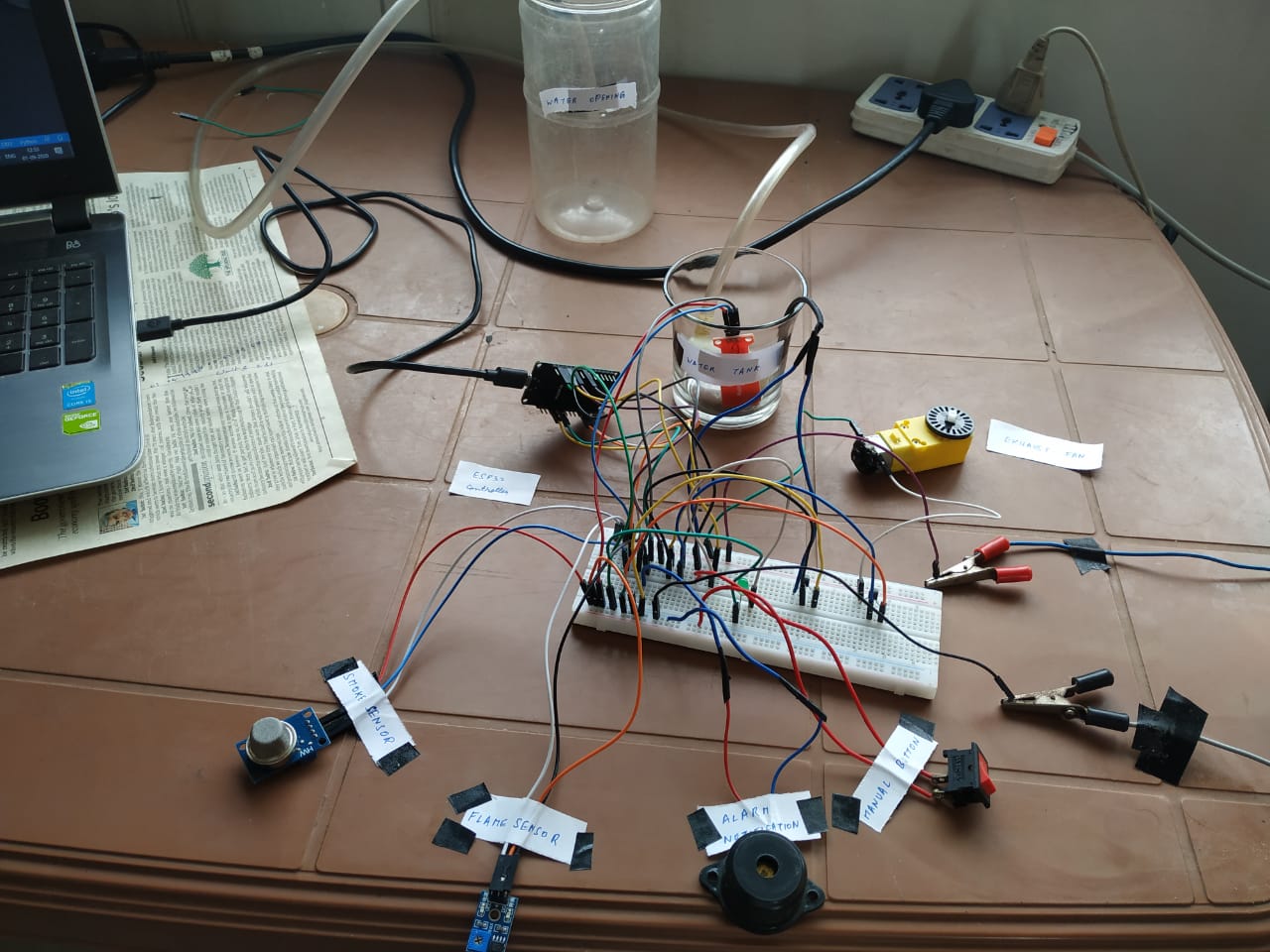


Figure 4. 14: Hardware setup

The figure 4.14 shows the hardware setup for proposed system, based on proposed design. The arrangement of the hardware based on the system requirement. ESP32 controller is used to fetch the data from different sensors, performs the evaluation and makes the decision when to actuate things. The output is seen in the Serial monitor for C++ and REPL console for Micropython.

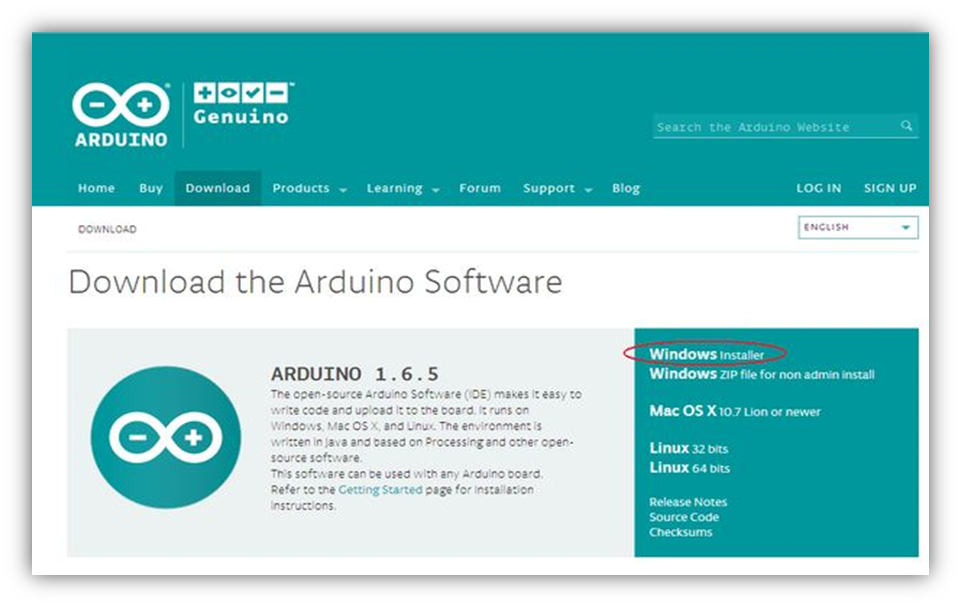
**Figure 4. 15 Hardware setup of System developed**

### 4.5.2 Software Setup:

Software set up is a part of the developing process to getthe software part of the system. There are two different software used for developing the system. Arduino IDE is used to dumb the C++ code in ESP32 Dev module. Setting up Micropython to ESP32 is totally a different process. Visual Studio Code is used to dump the MicroPython code to ESP32 Dev module.

#### 4.5.2.1 Setting up ESP32 in Arduino IDE:

**Step 1: Downloading and installing the Arduino IDE**

****Figure 4. 15: Installing IDE

**Step 2: find the preference dialog.**

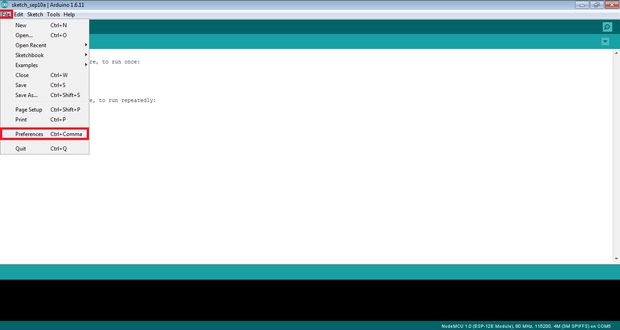
First open the Arduino IDE and go to file menu select the preference click on it. After that it will open the preference dialog box it is shown in below figure 4.12.

Figure 4. 16: Preference Dialog box

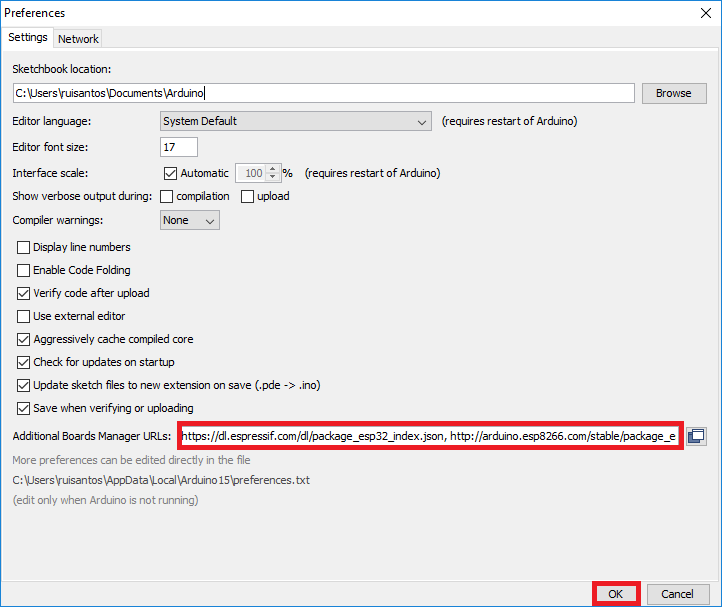


Figure 4. 17: ESP32 data package

In this figure 4.17 preference dialog box we have to need give URL of the ESP32 data package path.

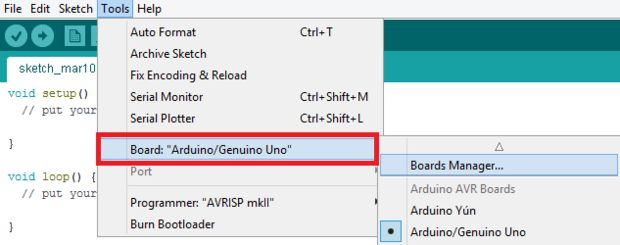
**Step 3: where to find Board manager dialog**

Figure 4. 18: Board manager Dialog

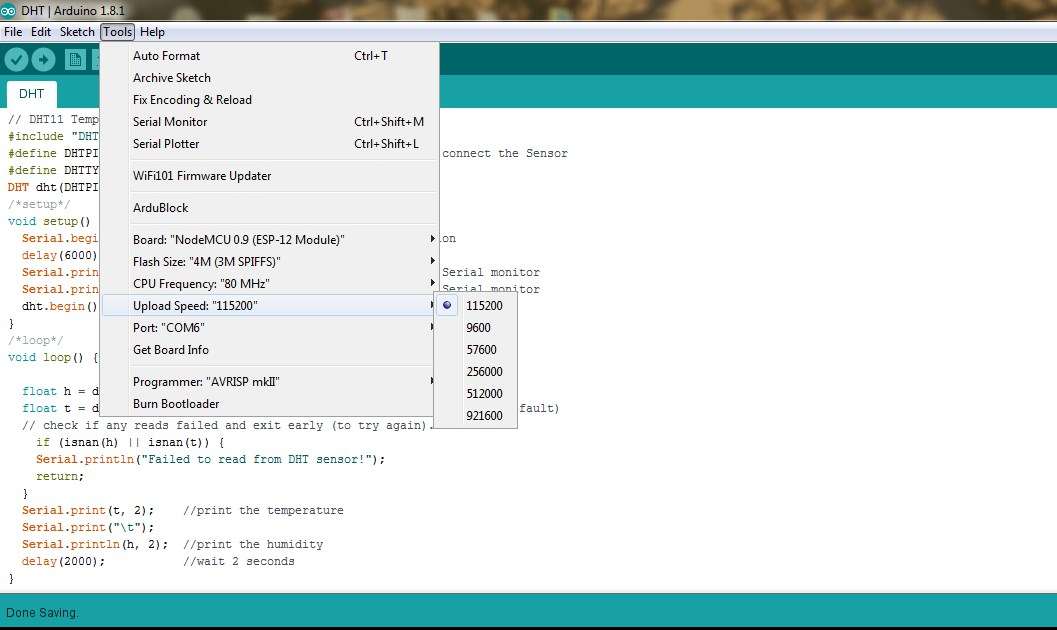
**Step 4. Determining COM port and configuring the Baud rate**

Figure 4. 19: configuring COM port Speed

**Step 5. Test Drive the Set up.**

Figure 4. 20: Testing the working

#### 4.5.2.2 Setting up Micropython to ESP32 board

There are few steps to be followed to get Micropython up and running

**Step 1. Install Python 3**

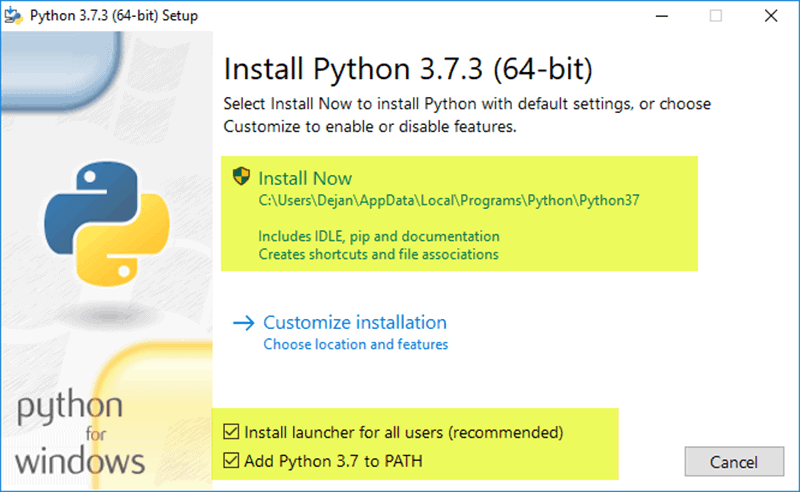


Figure 4. 21: Installing python



Figure 4. 22: Checking Python's working

**Step 2. Install esptool**

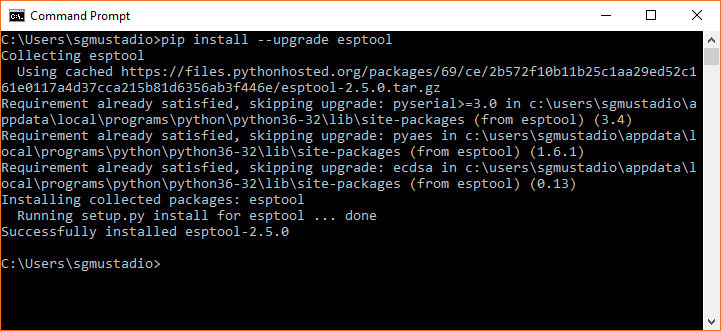


Figure 4. 23: Install esptool

**Step 3. Erase the Flash memory of ESP32**

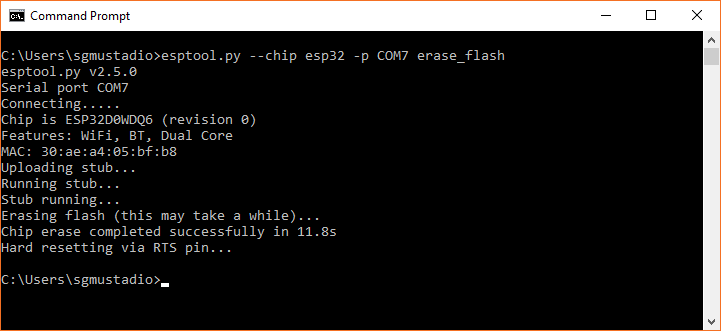


Figure 4. 24: Erase flash memory

**Step 4. Flash the MicroPython Firmware to the ESP32’s flash memory**

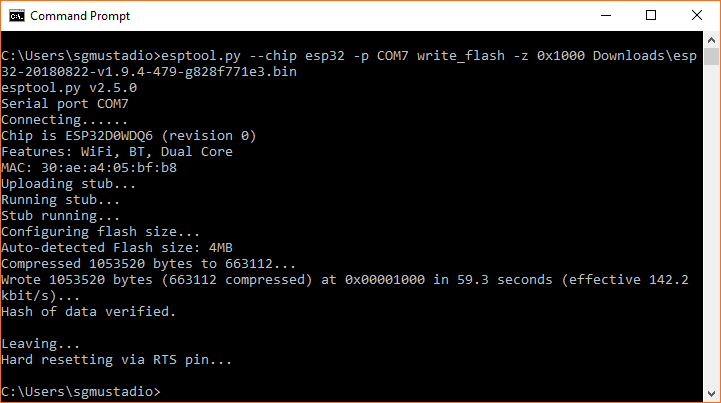


Figure 4. 25: Dump the Micropython firmware

**Step 5. Testing it**

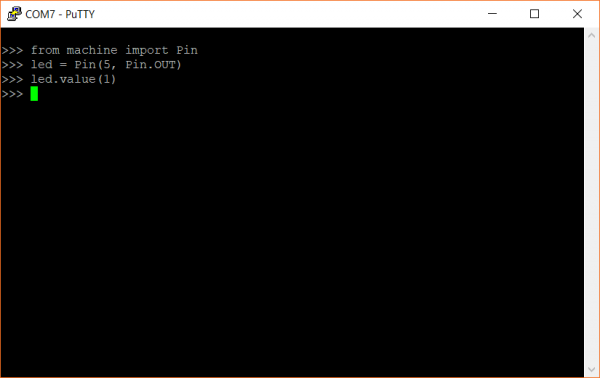


Figure 4. 26: Testing Micropython

#### 4.5.2.3 Setting up Visual Studio Code for MicroPython

To compile and run MicroPython programs on ESP32 using Visual Studio code IDE, it will just take two minutes to make it happen

**Step 1. Downloading Pymakr extension from Visual Studio Code**

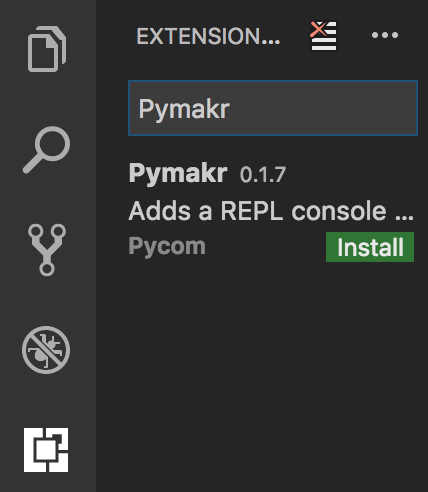


Figure 4. 27: Pymakr extension

**Step 2. Plugin the ESP32 and wait for the connection**

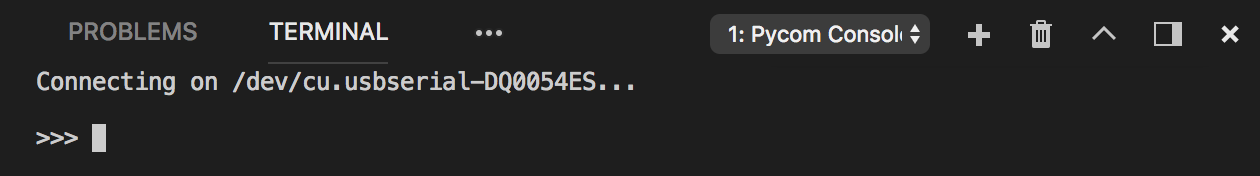


Figure 4. 28: Detecting COM port

**Step 3. Now you can run the MicroPython Program easily using Ctrl+ Alt + R.**

### 4.5.3 System Implementation

In this section of the report different software interfacing to the hardware system will be discussed. Same design, controller (ESP32) and platform is used for developing the Fire detection and Alarming system. Even the code structure is the same.

#### 4.5.3.1 Implementation between ESP32 and sequential C++ programming

C++ has been the platform for Embedded programming for a long time. This is a low level language with higher speed and easy to code.

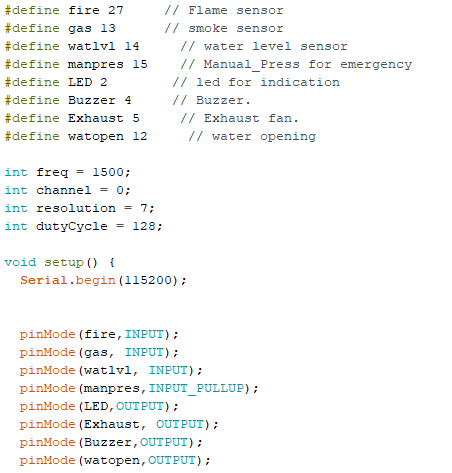


Figure 4. 29: Sequential C++ Code snippet 1

The above figure 4.29 shows the code snippet for which the pins of the sensors and actuators are connected to the ESP32 board. The frequency, channel and resolution variables are setup for giving the inputs to the buzzer.

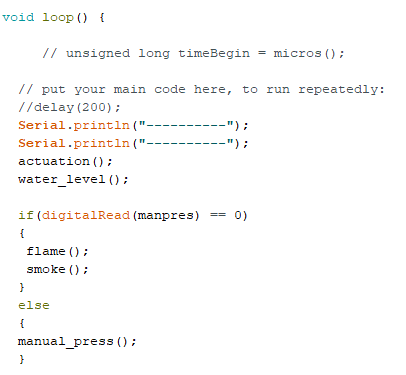


Figure 4. 30: Sequential C++ code snippet 2

The above figure 4.30 shows the loop function which is given in every Arduino IDE program this denotes that the things applied inside this runs in an infinite loop. The different sensor integrations are given in separate functions and all these functions are called inside the loop to make it run continuously.

#### 4.5.3.2 Implementation between ESP32 and multithreaded C++ programming

For adding real time capabilities to the C++ program FreeRTOS API is used. ESP32 has underlying FreeRTOS operating system.

The below figure 4.31 shows the different task handles given for different tasks. In real-time programming, tasks are nothing but threads. So, here for each task a task handle is given and semaphore is given for synchronization.

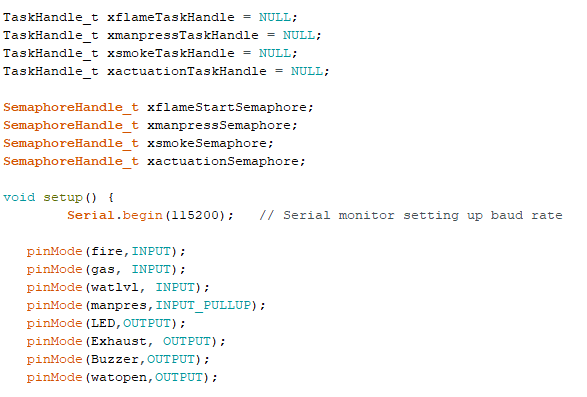


Figure 4. 31: Multithreaded C++ code snippet 1

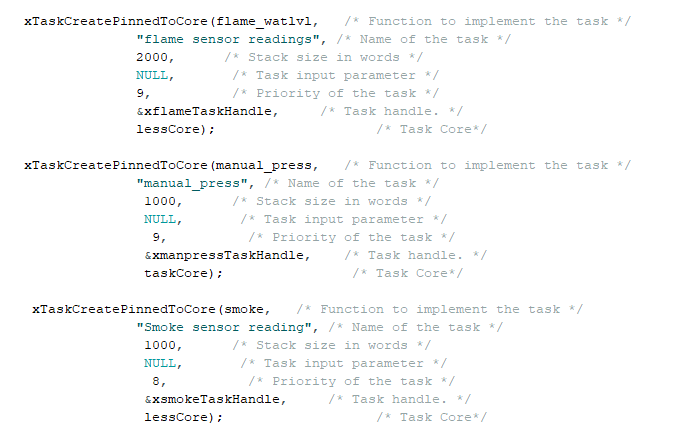


Figure 4. 32: Multithreaded C++ code snippet 2

The above figure 4.32 shows the different tasks that are created to perform different sensors and actuators work. Here task pinned to the core feature is used since ESP32 has 2 cores and we have 4 tasks here, so it’s distributed and assigned to 2 tasks per core. Regarding the priority level flame sensor, water level and manual press button is given as task priority 9, smoke sensor is given as task priority 8 so it runs after the flame sensor and manual press. The actuation task is given as 10 so it is given the highest priority.

#### 4.5.3.3 Implementation between ESP32 and sequential Micropython

Before performing the coding the Micropython firmware is dumped to the flash memory of the ESP32.



Figure 4. 33: Sequential Micropython code snippet 1

The above figure 4.33 shows the code snippet for sequential Micropython where the variables used are being initialized. The corresponding GPIO pins from the ESP32 are being set to the sensors and actuators.

Each sensors are given as separate functions and their readings are being read. All these functions are called back in a while loop to run it immediately. The import machine, esp, pin, adc libraries are inbuilt modules present in Micropython firmware and we will have to import them to use ESP32.

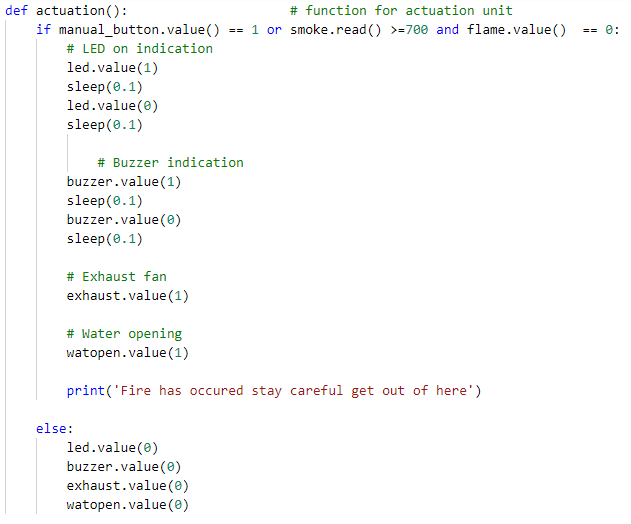


Figure 4. 34: Sequential Micropython code snippet 2

The above snippet shows the actuation function this contains four actuations LED, Buzzer, Exhaust fan and water opening to stop the fire in the else part it doesn’t implement anything.

#### 4.5.3.4 Implementation between ESP32 and Micropython with threads

The design is the same the only difference is the functions are given as separate threads and their values are tested.

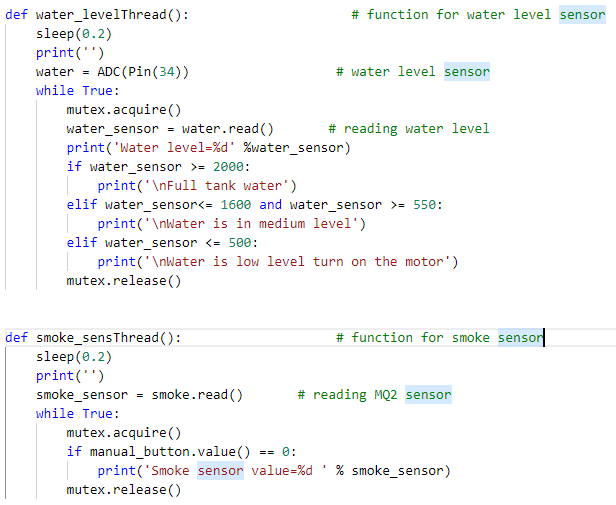


Figure 4. 35: Multithreaded micropython code snippet 1

The above figure 4.35 shows the thread that performs the water level sensing in the tank. So based on the voltage level the threshold has been set as low, high or medium level. Smoke sensor analog values are also printed in a real-time manner.

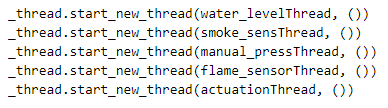


Figure 4. 36: Multithreaded Micropython code snippet 2

The above figure 4.36 shows the different thread implementation and each sensors and actuators are given as different threads. The threads run concurrently. Mutex synchronization mechanism is given used to get the systematic results. The \_thread library in Micropython is still under development and there might be some timing error or latencies.

## 4.6 System Integration:

After implementation is done on software now the performance analysis has to be performed based on the various parameters. For testing the latency the Oscilloscope and Function Generator were used.

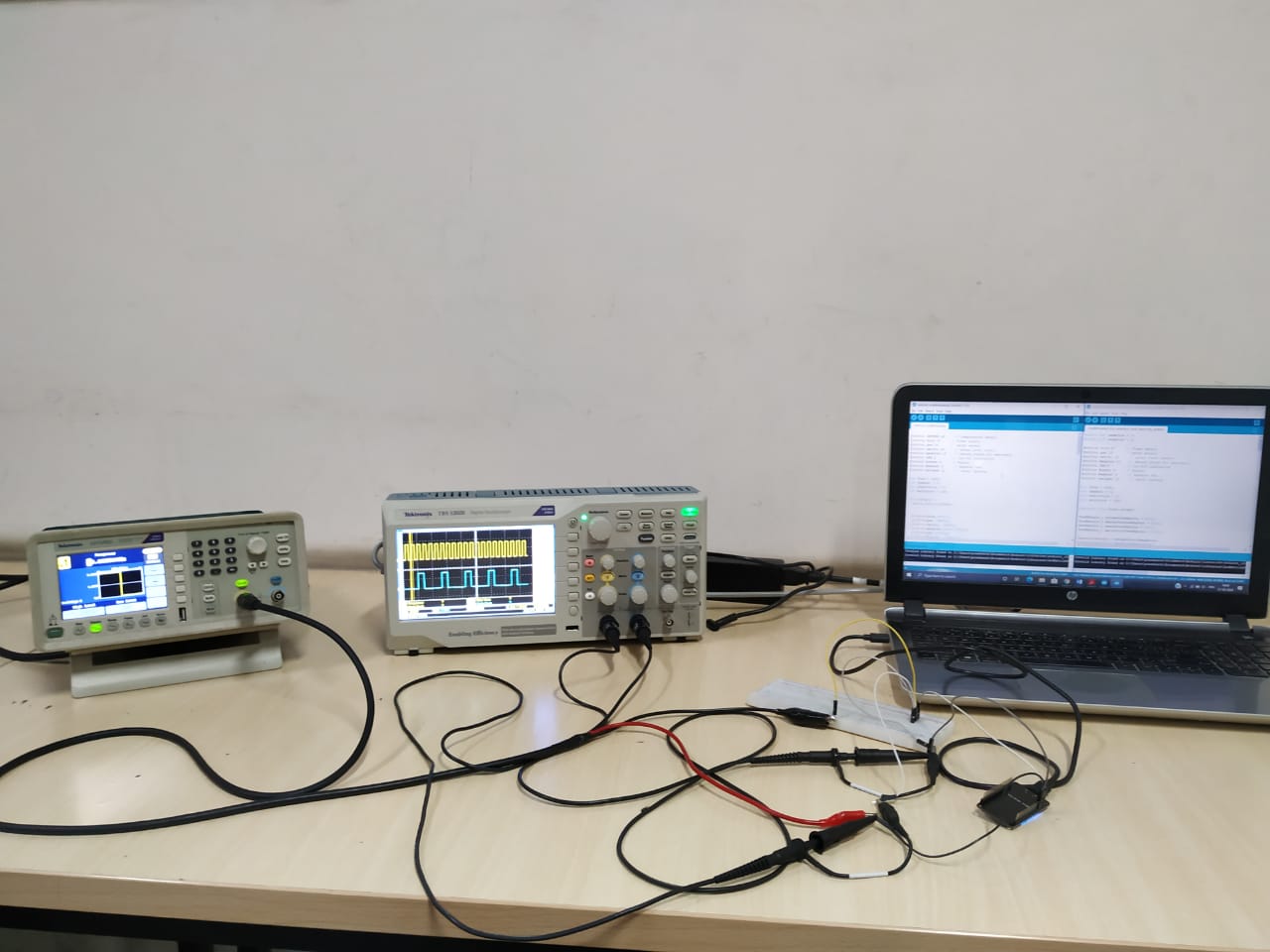


Figure 4. 37: Latency test setup

From the figure\_\_ you can notice that the input GPIO pin is given to the Function generator and output GPIO pin is given to the Oscilloscope. The latency readings are taken and the further proceedings are being committed.

# 5. Results and Discussions

In this chapter of dissertation work the testing part of developed system will be evaluated based on their working functionality. The Fire detection and Alarming system developed and implemented is tested based on unit testing, integration testing, system level testing and performance testing.

The performance analysis is being tested and evaluated using the devices such as CRO and Function generator. CRO shows the results such as expected latency and worst case latency.

All test cases developed discussed in this chapter and based on the results for each test cases the developed system tested at integration and system level.

## 5.1 Unit Testing

Unit testing is done with each individual component and testing the functionality of individual component for a system. It includes all round testing for that particular component.

In this testing it involves to test the functionalities of Smoke sensor(MQ2), Flame sensor(YS-17), Water level sensor, DC motor(exhaust fan) and Submersible pump in Arduino IDE(Sequential C++ and multithreaded C++) and Visual Studio Code(Micropython and multithreaded Micropython).

Table 5. 1: Unit testing test cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case number** | **Description** | **Test Process** | **Expected result** | **Actual result** | **Pass/Fail** |
| 1[Refer image 5.1] | Interfacing Smoke sensor with ESP32 | Smoke sensor is given as input to ESP32 and in the other end serial communication is established with ARDUINO IDE and REPL console with Visual Studio Code | Should show the analog value according to real time data | Giving immediate response for the change in analog values | PASS |
| 2[Refer image 5.2] | Interfacing flame sensor with ESP32 | Flame sensor is given as input to ESP32 and in the other end serial communication is established with ARDUINO IDE and REPL console with Visual Studio Code | Should show the digital value according to real time data | Giving immediate response for the change in digital values | PASS |
| 3[Refer image 5.3] | Interfacing Water level sensor with ESP32 | Water level sensor is given as input to ESP32 and in the other end serial communication is established with ARDUINO IDE and REPL console with Visual Studio Code | Should show the analog value according to real time data | Giving immediate response for the change in analog values | PASS |
| 4[Refer image 5.4] | Connecting a DC motor to ESP32. | A DC motor is connected to ESP32 and tested for ON and OFF cycles. serial communication is established with ARDUINO IDE and REPL console with Visual Studio Code. | Should turn ON and OFF for every given cycle delay. | The motor works fine and has instant ON and OFF switching. | PASS |
| 5[Refer image 5.5] | Connecting a Submersible motor to ESP32. | A Submersible motor is connected to ESP32 and tested for ON and OFF cycles. serial communication is established with ARDUINO IDE and REPL console with Visual Studio Code | Should turn ON and OFF for every given cycle delay | The motor works fine and has instant ON and OFF switching | PASS |

The unit testing is done to verify the functionality of the system develop with the simulation set up built for taking input value from the sensor of the system.



Figure 5.1: Smoke sensor interface with ESP32

The above figure 5.1 shows the experimental setup for smoke sensor analog reading sensing. The figure is the reference for test case 1.

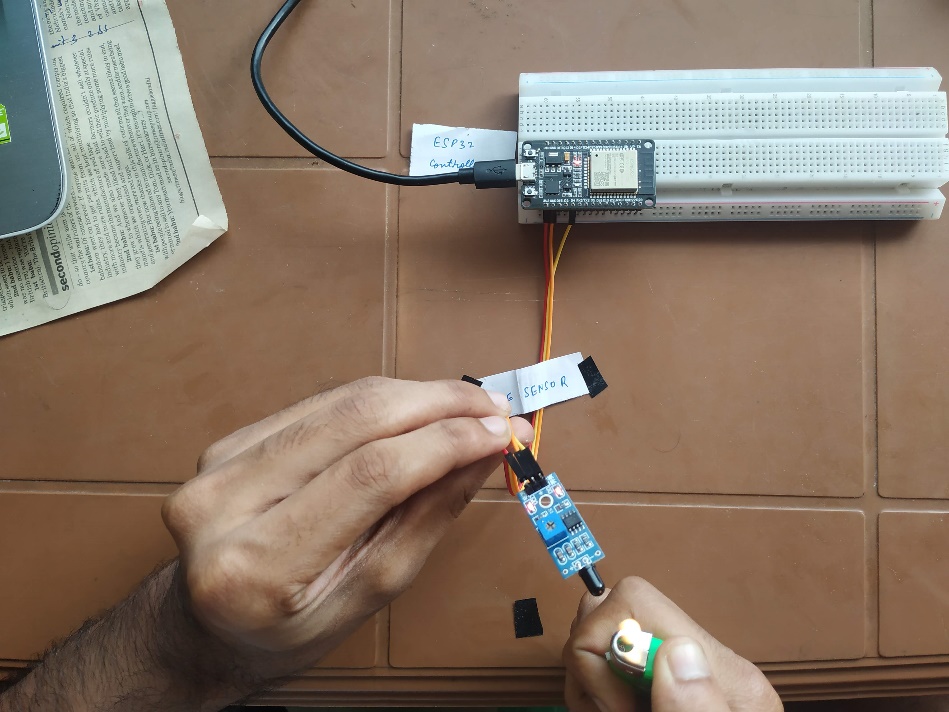


Figure 5. 2: Flame sensor integration with ESP32

The above figure 5.2 shows the flame sensor unit testing. This setup refers to the experimental setup test case 2.

The below figure 5.3 shows the water level sensor unit testing. This setup refers to the experimental setup test case 3.



Figure 5. 3: Water level sensor integration with ESP32

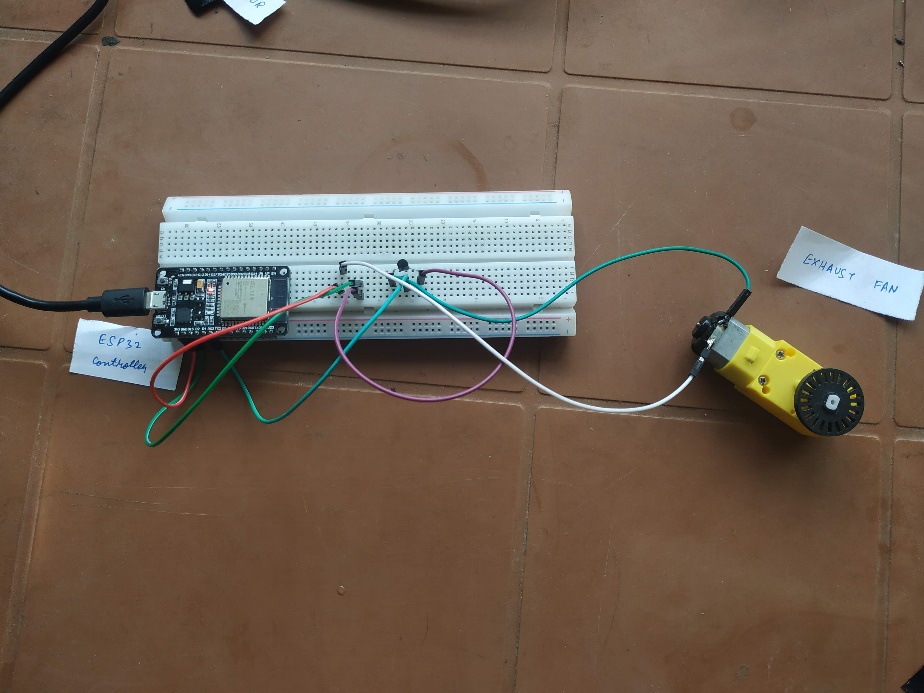


Figure 5. 4: DC motor integration with ESP32

The above figure 5.4 shows the exhaust fan unit testing. This setup refers to the experimental setup test case 4.

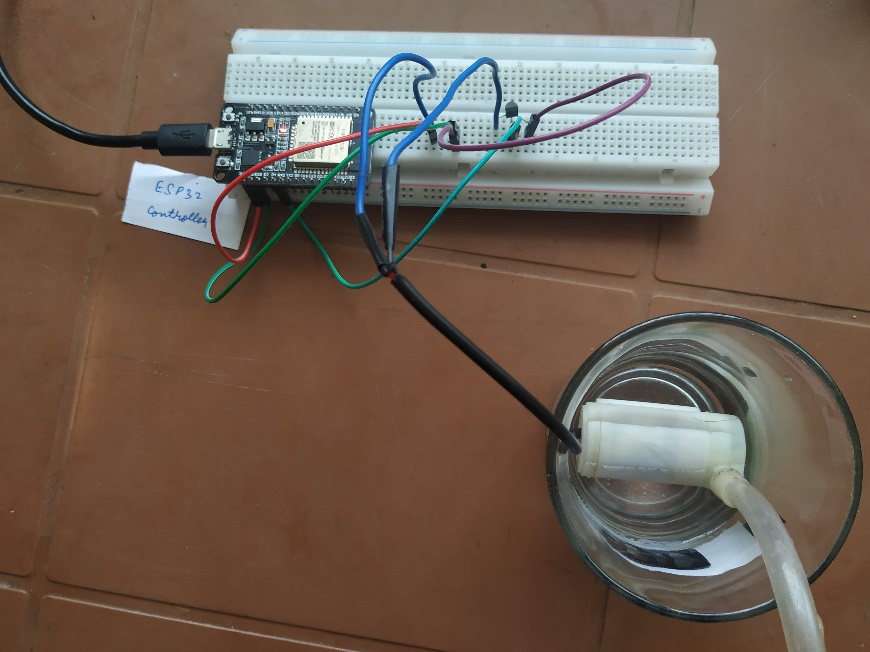


Figure 5. 5: Submersible water pump interface with ESP32

The above figure 5.5 shows the submersible water pump unit testing. This setup refers to the experimental setup test case 5.

## 5.2 Integration Testing

The integration testing is nothing but integrating two or three different sub system and testing for the working. For integration testing some test cases can be given as,

* To test for manual press switch with the Submersible pump and exhaust fan.
* To test for smoke, flame sensor and exhaust fan and testing for the functionality.
* To test for flame sensor, smoke sensor with LED and buzzer.
* To test for manual press button with LED and buzzer.

Table 5. 2: Integration testing test cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case number** | **Description** | **Input** | **Expected result** | **Actual result** | **Pass/Fail** |
| 1 | Integrating manual press switch, submersible pump and exhaust fan with ESP32 | Manual Press switch is the input | The submersible pump and motor should be ON when the switch is turned ON | Instant response from the system for ON and OFF | PASS |
| 2 | Integrating smoke sensor, flame sensor and exhaust fan with ESP32 | Smoke and flame sensor is the input | The exhaust fan should be ON when it goes higher than the set threshold of smoke and flame sensor. | Instant response from the system for ON and OFF | PASS |
| 3 | Integrating smoke sensor, flame sensor, LED and buzzer with ESP32 | Smoke and flame sensor is the input | The LED and buzzer should turn ON when it goes higher than the set threshold of smoke and flame sensor. | Instant response from the system for ON and OFF | PASS |
| 4 | Integrating manual press switch, LED and buzzer with ESP32 | Manual Press switch is the input | The LED and buzzer should turn ON when manual press switch is turned ON. | Instant response from the system for ON and OFF | PASS |

## 5.3 System level Testing

In system level testing the whole system will be tested for its functionality.

* To test for manual press switch actuation.
* To test for smoke and flame sensor actuation.
* To test for water level sensor readings irrespective of any actuation.

Table 5. 3: System level testing test cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case number** | **Description** | **Input** | **Expected result** | **Actual result** | **Pass/Fail** |
| 1 | To test for Manual Press switch actuation | Manual Press switch | When the manual press button is ON all the actuation like water opening, exhaust fan, LED and buzzer should be ON | System actuation successful | PASS |
| 2 | To test for smoke and flame sensor actuation | Smoke and flame sensor | When the smoke sensor goes beyond set threshold and when flame is detected all the actuation like water opening, exhaust fan, LED and buzzer should be ON | System actuation successful | PASS |
| 3 | To test for water level sensor readings irrespective of any actuations | Water level sensor | The water level sensor should be ON always and it should measure the water level irrespective of any actuation too. | Water level sensor works perfectly irrespective of any actuations | PASS |

## 5.4 Result Analysis

Based on the addressed problem identification and the requirement analysis done on the research gaps. The system proposed with some additional solutions. After implementing the system on hardware, Results for the developed system are shown in this section of the thesis. Results are obtained from different languages like C++ and Micropython(both sequential and multithreaded).

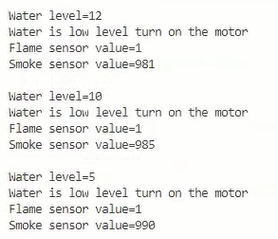


Figure 5. 6: Default result monitor screen for Micropython

The figure 5.6 shows the output screen when there is no fire and manual press button isn’t ON. By default all the sensor inputs are given to the ESP32 all the time. The output screen represents for Micropython.

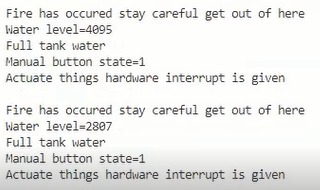


Figure 5. 7: Result monitor screen after manual interrupt is pressed in Micropython

The figure 5.7 shows the output screenshot when the manual interrupt has been triggered. All the actuation takes place. Irrespective of any actuation water level sensor senses for the level of water in the tank. The output screen represents for Micropython.

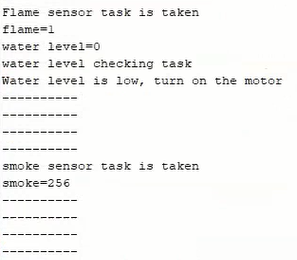


Figure 5. 8: Default output screen C++

The figure 5.8 shows the output screenshot when there is no fire and all the sensor provides the input to the ESP32 to process the data. The output screen corresponds to C++ when the manual press is OFF.

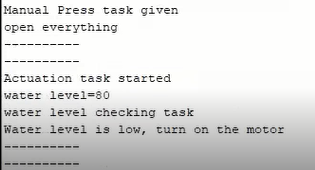


Figure 5. 9: Manual press triggered actuation output screen C++

The figure 5.9 shows the output screenshot when the manual interrupt is triggered. Irrespective of any actuation water level sensor senses for the level of water and if the water level is low in the tank it sends instruction to turn on the motor. The output screen corresponds to C++ when the manual press is ON.

### 5.4.1 Performance Analysis:

Performance analysis is done to make sure to analyze the behavior of two different languages C++ and Micropython [both sequential and multithreaded]. The developed system is analyzed for the following parameters latency, worst case response time, Jitter, execution time, heap memory allocated, correctness. The tests were done under different functionalities,

Minimum functionality- Just a small program sensor and LED toggle.

Maximum functionality- The whole Fire detection and Alarming program

#### 5.4.1.1 Latency:

Latency may be considered as the time difference between the moment that an interrupt happens and the moment that a response is generated from the associated interrupt handler. Nearly 60 readings were taken in different frequency of 1 Khz to 30 Khz and average was taken to conclude this result.

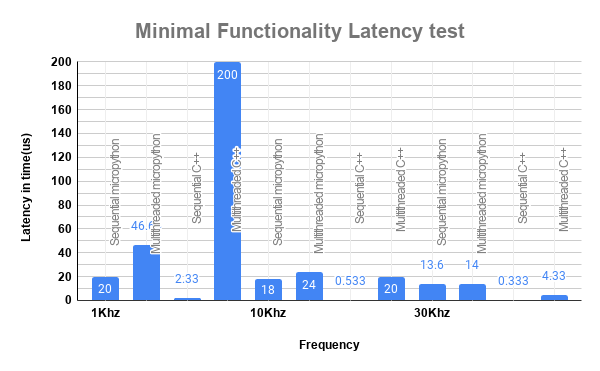


Figure 5. 10: Minimal load latency test

The above figure 5.10 shows the performance analysis for latency in minimum functionality (just one button and led toggle). This is compared around 4 different programming techniques. Sequential C++ programming out performs all the other techniques by having very least latency. When there is minimal functionality sequential C++ out performs every other language techniques.

Multithreaded C++ shows a huge spike in the latency since there is very minimal load and it is trying to get into the FreeRTOS and fetch the data. Since, there is very minimal instructions sequential C++ is able to achieve it.

The below figure 5.11 shows the best reading of sequential C++ with minimum load which is 400ns.

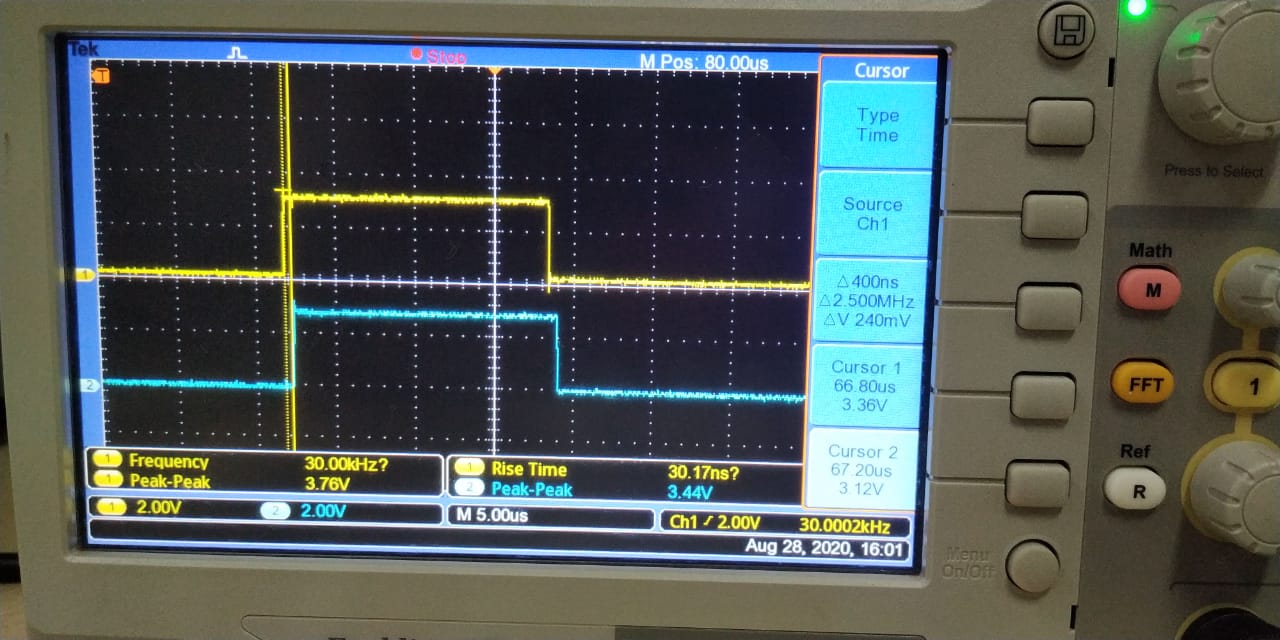


Figure 5. 11: Sequential C++ CRO screen

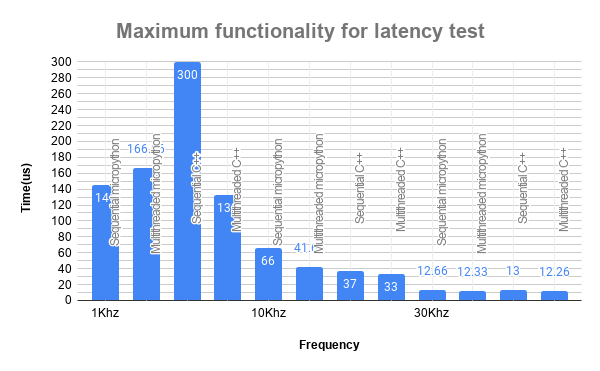


Figure 5. 12: Maximum load for latency test

The above figure 5.12 shows the latency of 4 different language techniques for different frequency in Maximum functionality condition. Here maximum functionality condition is when the whole program is dumped to the controller. The readings were concluded by taking 45 different samples and averaging the value. Multithreaded C++ proves to be having less latency when compare to other techniques.

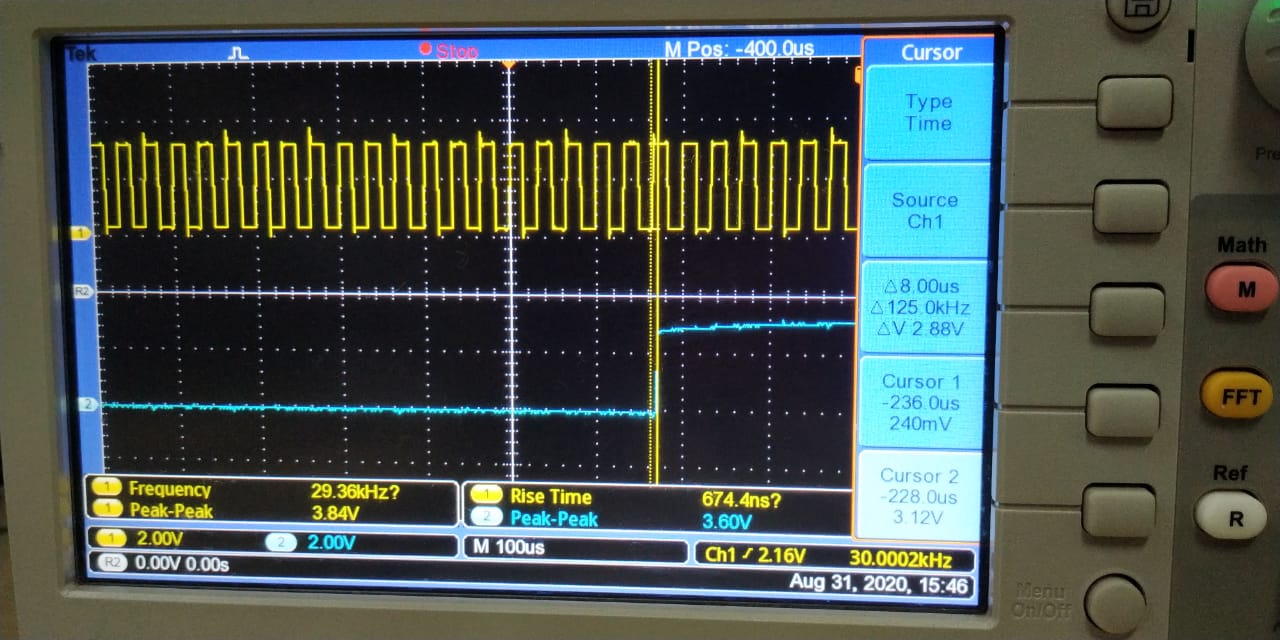


Figure 5. 13: Multithreaded C++ CRO screen

The below figure 5.13 shows the best latency reading of multithreaded C++ achieved using FreeRTOS in maximum load which is 8us.

#### 5.4.1.2 Worst case response time:

WCRT may be described as the inverse of the maximum interruption frequency with reliability. In order to obtain such parameter, the input signal frequency from the function generator should be slightly incremented and the output signal should start changing. Then, the maximum interruption frequency that is readable on the oscilloscope screen with reliability is used to calculate the WCRT. In total, 60 independent samples of measurement were taken. The worst case latency in minimal and maximum load will be discussed.

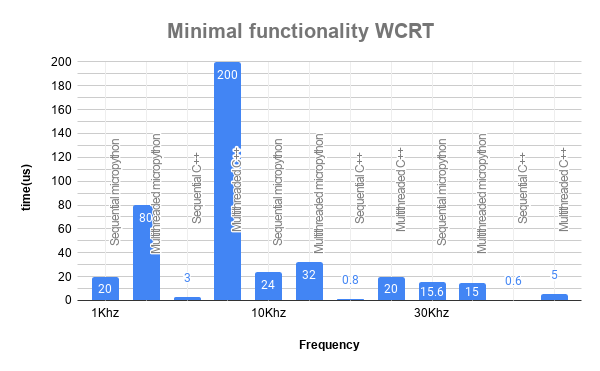


Figure 5. 14: Minimal load WCRT

The above figure 5.14 shows the worst case response time for different programming techniques. The readings are taken for different frequencies and an average is taken. The test is done under minimal load and the sequential C++ has very less WCRT. In minimal functionality always sequential C++ performs well. But in 5 Hz and 10 Hz multithreaded C++ and Micropython has more WCRT.

The below figure 5.15 shows the WCRT in the maximum loaded condition. During maximum loaded condition it is pretty evident that multithreaded C++ has the best WCRT and it outperforms other techniques. In maximum functionality always multithreaded C++ performs well.



Figure 5. 15: Maximum load WCRT

#### 5.4.1.3 Jitter:

Jitter may be considered as a random variation between each latency value and is obtained from several latency measurements. The jitter is a parameter that can cause a notorious impact in a RTOS. In order to figure out the jitter, the time variance between two consecutive latency measurements is calculated. The greatest value encountered is pointed out as the worst jitter of the system. Jitter is measured under maximum functionality condition.

The below figure 5.16 shows the jitter for different variations of C++ and Micropython. Jitter is tested in maximum functionality scenario where the whole program is dumped to the controller. The Jitter value here is the difference between two consecutive latency readings and the greatest value is pointed out as jitter.

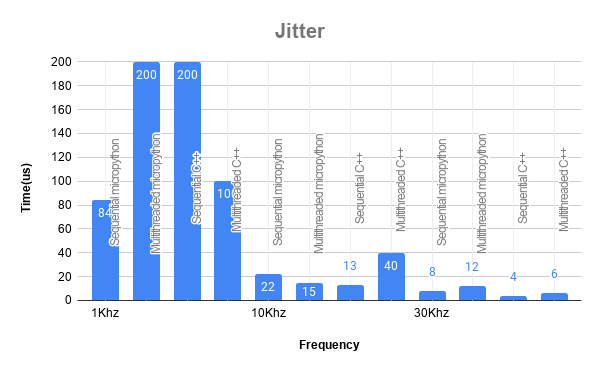


Figure 5. 16: Jitter

#### 5.4.1.4 Execution time:

The total time for the program to complete its operation is considered as the execution time. It depends from programming language to language. Execution time is measured under maximum functionality condition.

The below figure 5.17 shows the total execution of the program(this is for the whole program). Multithreaded C++ takes just 171ms to complete the cycle of execution whereas sequential Micropython takes around 470ms. Even though the latency for the Micropython is lesser than sequential C++ it takes more time to execute because Python is an interpreter and high level language whereas C++ is a compiler and low level language. So, there will definitely be latency when whole process is executing since the interpreter checks for each line of code individually whereas compiler compiles whole code.

Python can’t compete with C++ in execution speed but python is considered only for ease of programming and perfect time to market.

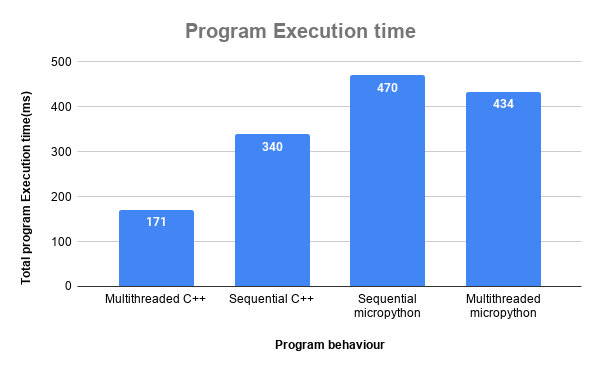


Figure 5. 17: Execution time

#### 5.4.1.5 Allocated heap memory:

Heap memory is the memory used by the system during the process of program running. A memory heap is a location in memory where memory may be allocated at random access. Unlike the stack where memory is allocated and released in a defined order, individual data elements allocated on the heap are typically released in ways which is asynchronous from one another. Allocated heap memory is measured under maximum functionality condition.

The below figure 5.18 depicts the heap memory consumed by each code. Micropython code is smaller, because lots of code which is needed is part of the micropython engine, while for C++ programs it is either part of your code or linked with your code. [This includes I/O and data handling.]

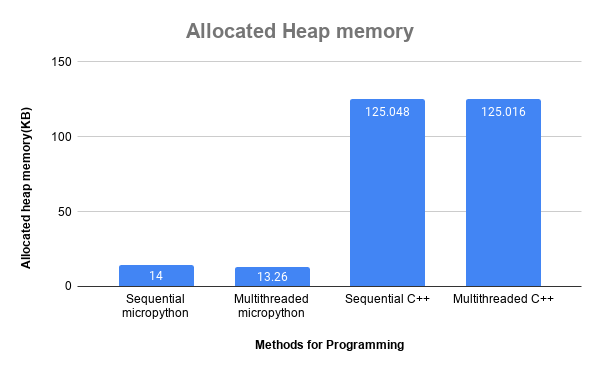


Figure 5.18: Allocated Heap memory

### 5.4.2 Existing system vs Proposed system

The Existing system [de Oliveira Turci, Luca., 2017] was developed using Arduino controller and FreeRTOS coding. The components used were smoke, flame sensor, exhaust fan, alarming unit (LED and Buzzer). The lowest latency achieved was 12us.

But, the proposed system includes Water opening (Submersible water pump), Manual press button and water level sensor to monitor the tank level in addition to the Existing system. The latency for the maximum functionality is about 12.26us and multithreaded C++ achieves it with ease. The multithreaded Micropython wasn’t able to compete with multithreaded C++ and it’s still under development.

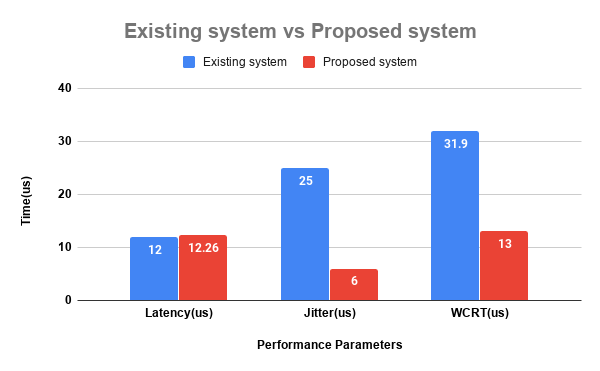


Figure 5.19: Existing system vs Proposed system

The figure 5.19 depicts the comparison of existing system to the proposed system. They are compared for 3 different parameters Jitter, Latency and WCRT. The results are for maximum functionality at 30 Khz frequency for multithreaded C++ programming

# 6. Conclusions and Future Directions

## 6.1 Summary of Work

A Fire detection and Alarm system was designed and developed using different categories of programming languages. Till now the major programming languages we have been using for development of Embedded systems is C/C++ but in this analysis the major competitor will be participating which is Python compatible with controllers and a smaller version Micropython was considered.

It can be concluded from obtained result that, C++ perform well when compared with Micropython. This is because of 2 reasons

* C++ is low level language whereas Python is high level language.
* C++ is a compiler language whereas Micropython is interpreter language. So interpreter language always takes time to execute.

## 6.2 Conclusion

* Smoke sensor, Flame sensor, water level sensor and manual press button is given as input to the ESP32.
* ESP32 processes the data and make the actuators [water opening, Exhaust fan, Light indication and Alarm] to perform.
* Latency, WCRT and Jitter calculation is tested on 3 different functionalities with low, medium and maximum to give the final figure.
* The execution time and heap memory are calculated from the function of the firmware.
* Multithreaded C++ outperforms all others techniques by it total program execution which is very instant in Maximum functionality testing. This makes RTOS and C++ language very unique.

## 6.3 Future Work

It is important aspect to consider the future expansion of any developed work and to concentrate on future scope of the implemented work. Some future extensions can be considered for the Performance Analysis of a concurrent Fire detection and Alarming system as,

* The hard real- time application can be developed using different languages like Lua, Java, Java script, Go, Ada etc and performance analysis can be done.
* The system can be tested for Synchronization complexity which can added up for comparison of performance analysis.

# References

1. Higuera, T. (2012). About 15 years of Real-Time Java. Madrid 28040, Spain: Universidad Complutense de Madrid.
2. Ray, B., Posnett, D., Devanbu, P. and Filkov, V. (2017). A large-scale study of programming languages and code quality in GitHub. Communications of the ACM, 60(10), pp.91-100.
3. Zain, F., Tobias, M., Olaf, G., Andreas, L. (2019). Event-Driven Multithreading Execution Platform for Real-Time On-Board Software Systems.
4. Oyenike, B. (2012). COMPARATIVE ANALYSIS OF SOME PROGRAMMING LANGUAGES. Transnational Journal of Science and Technology, [online] 2(5), pp.107-118.
5. Sreeram, V., Surya, M., Akash, G., Niranjan, R., Sree, B., Shruthi, B. and Mohamed, S. (2018). Realtime Applications with RTMaps and Bluebox 2.0. International Conference of Artificial Intelligence 2018.
6. Nanz, S., Torshizi, F., Pedroni, m. and Meyer, B. (2013). Design of an empirical study for comparing the usability of concurrent programming languages. Information and SoftwareTechnology, 55, pp.1304-1315.
7. Nanz, S. and Faria, C. (2015). A Comparative Study of Programming Languages in Rosetta Code. 2015 IEEE/ACM 37th IEEE International Conference on Software, [online] pp.778-788.
8. Sheikh, A., Ghazala, I., Noman, K. (2016). A qualitative study of major programming languages. International Journal of Information and Communication Technology.
9. Silvia, D. (2015). The role of concurrency in an evolutionary view of programming abstractions. arXiv:1507.07719v1 [cs.PL] 28 Jul 2015
10. de Oliveira Turci, Luca. (2017). Real-Time Operating System FreeRTOS Application for Fire Alarm Project in Reduced Scale. International Journal of Computing and Digital Systemss. 6. 198-204. 10.12785/IJCDS/060405.
11. Mahmoud, A., Rabbah, N., Rabbah, H., Mounir, R. (2018). Python in Real Time Application for Mobile Robot . Smart Application and Data Analysis for Smart Cities (SADASC'18).
12. David, A., Christophe, D., Valerie, P., Fatiha, B., Maxime, B., Luc, Legres., Anne, J., Philippe, B. and Jean, Y. (2014). Automatic Image Quality Assessment in Digital Pathology: From Idea to Implementation. Sorbonne Paris Cité, 75205 Paris Cedex 13, France
13. Kulkarni, S., Prasanna, T., and Bramaramba, K. (2019). An IoT based Fire Detection, Precaution & Monitoring System using Raspberry Pi3 & GSM. INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 08, Issue 07 (July 2019)
14. Rafael, V. and Glauco, C. (2010). Real Time Operating System (RTOS) Comparison. Universidade de São Paulo (USP), São Carlos, Brazil.