

# Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATMega32A using FreeRTOS



By

Abdelrahman Maher Abdelhamid 186341

A Dissertation submitted to the Faculty of Engineering, The British University in Egypt, in  
Partial Fulfillment of the requirements for the bachelor's degree in electrical engineering

Supervisor: Dr. Amr Ismail Zamel

June, 2023

Electrical and Communication Engineering Department

Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on  
ATMega32A using FreeRTOS

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATMega32A using FreeRTOS

Dissertation Approved:

Dr. Amr Ismail Zamel

---

Committee member

---

Committee member

---

External committee member

Affiliation:

---

---

Dean Faculty of Engineering

---

## Acknowledgments

I want to sincerely thank everyone who helped me with my research project and for their support. I want to start by expressing my gratitude to my supervisor for his advice, support, and priceless feedback. his knowledge and point of view have been essential in guiding my study and assisting me in completing my objectives.

I also want to express my gratitude to my family for their constant love and support. Their support and confidence in me has worked as a constant source of motivation and inspiration. Sincere thanks go out to them for their patience and sacrifices at this period.

Finally, I want to thank my friends for their encouragement and support during my study process. Their encouraging comments and infectious optimism have added to the significance of this event. I want to express my sincere gratitude to everyone who helped make the project a success.

## Abstract

This research paper discuss the Real Time Operating System ( RTOS ) and their architecture, types either hard or soft RT, software types, benefits, and drawbacks, as well as how they work, including task states and control blocks. This research goes into great detail to explain many scheduling algorithms, including Round Robin, Priority Based, Least Slack Time First, Earliest Deadline First, First Come First Serve, and non-preemptive Deadline First. It also covers the performance monitoring tools and what should be monitored while measuring the performance. The problem that the authors shed light on is the increment of death and incidents of Liquefied Petroleum Gas (LPG) burns and the patients within 21 to 60 experienced LPG burns the most frequently (73.85%) and almost 81.03% of this percentage was related to gas leak and neglecting the kitchen (2%). So, the author made an application by using FreeRTOS with different scheduling algorithms PB, RR “ with 3 different time slicing 1/10 , 1/100 , 1/10000 ” and PBRR that is implemented on ATmega32A which contain a couple of sensors with a lcd to display the sensors readings and take the appropriate response by using dc motor works as ventilating fan. The author shows the PPM values of CO and the time before critical symptoms begins. The results shows that the best scheduling algorithm in time execution was PBRR with time = 0.0656 sec while the rest takes more time than that and without RTOS takes 4.581sec however, the memory consumption of the application without RTOS was RAM = 163 Bytes which means 8 % full, ROM = 6112 Bytes which means 18.7% while the PBRR memory consumption was RAM = 1840 Bytes which means 89.9% full, ROM = 10316 Bytes which means 31.5% also there is a simulation that shows the execution of the tasks made by using Simso for PB scheduling algorithm only.

## Table of Contents

Acknowledgments .....	ii
Abstract.....	iii
List of Figures.....	vii
List of Abbreviations .....	ix
Chapter 1: Introduction.....	1
1.1. RTOS Architectures.....	2
1.2. Several Types of Software's for RTOS .....	2
1.3. Problem Statement .....	3
1.4. Research Objectives.....	3
1.5. Research Questions.....	4
1.6. Research Methodology .....	4
Chapter 2: Background .....	5
2.1. Types of RTOS .....	5
2.2. Components of RTOS.....	6
2.3. Characteristics of RTOS .....	7
2.4. Advantages of RTOS .....	8
2.5. Disadvantages of RTOS.....	9
Chapter 3: Literature Review.....	10
3.1. RTOS Applications .....	10
3.2. Scheduling Algorithm Updates.....	17
3.3. RTOS Developments. ....	19
Chapter 4: RTOS Operation .....	23
4.1. Task States .....	23
4.2. Task Control Block.....	24
4.3. Scheduling Algorithms .....	24
4.3.1. Shortest Remaining Job First .....	27
4.3.2. Round Robin (RR) .....	27
4.3.3. First Come First Serve (FCFS).....	27
4.3.4. Shortest Job First (SJF) .....	27

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

4.3.5. Deadline Monotonic (DM) .....	28
4.3.6. Rate Monotonic (RM) .....	28
4.3.7. Earliest Deadline First (EDF) .....	29
4.3.8. Least Slack Time First (LST) .....	30
4.4. Semaphore .....	30
4.5. Deadlock .....	31
Chapter 5: Proposed Performance Measurement on Different Scheduling Algorithms .....	32
5.1. Criteria of the Performance that May be Evaluated .....	32
5.2. Software to Measure the Performance .....	32
5.2.1. SEGGER System View .....	32
5.2.2. Percepio Tracealyzer .....	35
5.2.3. SimSo Simulator .....	35
5.1.4. Keil MDK by ARM .....	35
5.3. Analytical Method to Measure the Performance .....	36
5.3.1. Atmel ICE .....	36
5.3.2. By using the RTOS APIs .....	37
5.3.3. By using the Build in Timer .....	37
Chapter 6: Case Study Kitchen Air Control System. ....	40
6.1. The Components used and The Method .....	41
6.1.1. The MicroController .....	43
6.1.2. Temperature Sensor .....	44
6.1.3. Gas Sensor .....	44
6.1.4. LCD .....	47
6.1.5. DC Motor .....	48
6.1.6. Motor Driver .....	48
6.2. Layering Software Architecture .....	49
6.3. Implementing FreeRTOS on the Application .....	50
6.4. Creating Task in FreeRTOS .....	50
6.4.2. Putting the Task in the Block State .....	52
6.4.3. Putting the Task in the Suspended State .....	52
6.5. Implementing Different Scheduling Algorithms .....	52
6.5.1. Priority Based .....	52

# Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

6.5.2. Round Robin .....	53
6.5.3. Priority Based Round Robin .....	53
6.6. Hardware Implementation .....	54
6.7. Debugger.....	54
Chapter 7: Results.....	57
7.1. Simulation on Proteus .....	57
7.2. Debugging the Hardware .....	61
7.3. Simulating by using Simso .....	69
7.4. Segger SystemView .....	69
Chapter 8: Conclusion .....	70
References .....	72
Appendix A .....	78
Abstract .....	78
Problem Statement.....	78
Objective.....	79
Brief Background.....	79
Literature review .....	83
Methodology .....	85
Experimental or simulation work .....	86
Main results.....	87
Conclusion .....	87
References.....	88

## List of Figures

Figure 1-1: Embedded Systems Layers. ....	1
Figure 2-1: Soft RT system. ....	5
Figure 2-2: Hard RT system. ....	5
Figure 2-3: Components of RTOS. ....	7
Figure 3-1: System Block Diagram [8]. ....	10
Figure 3-2: System Block Diagram [9] ....	11
Figure 3-3: System Block Diagram [10]. ....	12
Figure 3-4: Block Diagram of the Chicken Poultry Farm Application [12]. ....	13
Figure 3-5: Block Diagram of Air Quality Monitoring System [13]. ....	13
Figure 3-6: Block Diagram of the Application [14]. ....	14
Figure 3-7: Block Diagram of Remotely Monitoring System [17]. ....	15
Figure 3-8: Two Possible Queuing Configurations for Tasks [23]. ....	18
Figure 3-9: Flowchart of the Proposed PI-RR Scheduling Algorithms [26]. ....	19
Figure 4-1: Full Task State Machine. ....	23
Figure 4-2: Task Control Block. ....	24
Figure 4-3: Scheduling Algorithm Classifications. ....	26
Figure 4-4: RR Tasks in the Queue ....	27
Figure 4-5: The SJF Algorithm's Flowchart ....	28
Figure 4-6: The Earliest Deadline First (EDF) Algorithm's Flowchart. ....	29
Figure 4-7: The LST Algorithm's Flowchart. ....	30
Figure 4-8: Deadlock Explanation Graph. ....	31
Figure 5-1: SEGGER SystemView Working Flow Diagram. ....	33
Figure 5-2: Continuous Recording Flow Diagram [46]. ....	34
Figure 5-3: Single Shot Recording Flow Diagram [46]. ....	34
Figure 5-4: Layering of Keil MDK [49]. ....	35
Figure 5-5: Atmel ICE. ....	36
Figure 5-6: ATMEL ICE Board Connection with Adapter Board ....	36
Figure 5-7: Flow Diagram of the APIs Method to Get the Task Execution Time. ....	37
Figure 5-8: Flow Diagram of the Timers Method to Get the Task Execution Time. ....	39
Figure 6-1: Block diagram of the Implemented Application. ....	42
Figure 6-2 :Flow Chart of the Implemented Application. ....	43
Figure 6-3: MQ2 Shape and Internal Structure [52]. ....	45
Figure 6-4: Sensing Element of MQ2 Sensor [52]. ....	45
Figure 6-5: Sensitivity Characteristics of the MQ2. ....	46
Figure 6-6: PWM Percentage Variations [57]. ....	48
Figure 6-7: Layering Architecture. ....	49
Figure 6-8: Layering of Software Implemented in the Application. ....	49
Figure 6-9: Chosen Files in Implementing the Code. ....	50
Figure 6-10: AVR JTAG pins. ....	55
Figure 6-11: JTAG Connection with Multiple Devices [60]. ....	55
Figure 7-1: Proteus Simulation while the Temperature more than 30. ....	57



Figure 7-2: Proteus Simulation while the Temperature less than 30.....	58
Figure 7-3: PWM Signal 50%. ....	58
Figure 7-4: PB Task Execution on Proteus. ....	59
Figure 7-5: RR Task Execution with Time Slicing (1/10) on Proteus. ....	59
Figure 7-6: RR Task Execution with Time Slicing (1/100) on Proteus. ....	60
Figure 7-7: RR Task Execution with Time Slicing (1/10000) on Proteus. ....	60
Figure 7-8: PBRR Scheduling Algorithm Execution Time.....	61
Figure 7-9: Assembly Conversion Provided by the Debugger.....	62
Figure 7-10: FreeRTOS Extensions on Atmel Studio.....	62
Figure 7-11: Task Handlers and TCB of Each Task. ....	63
Figure 7-12: Registers and their Bits.....	64
Figure 7-13: Simulation Shows the Task Execution Based on PB Scheduling Algorithm.....	69
Figure 7-14: Segger SystemView Implementation with the Application.....	69

## List of Tables

Table 1-1: Main Differences between RTOS and GPOS [2]. ....	1
Table 1-2: Different Architectures of RTOS. [2] .....	2
Table 1-3: Several Types of RTOS's Software's.....	2
Table 2-1: Different types of RTOS.....	6
Table 4-1: Preemptive and Non-Preemptive Scheduling [38, 24, 39, 27].....	25
Table 4-2: Static and Dynamic Scheduling. ....	26
Table 6-1: Comparison Between PB and RR Scheduling Algorithms [50]. ....	40
Table 6-2: PPM Values of CO and the Time Before Critical Symptoms [55, 56, 54].....	47
Table 6-3: Task Number and their Periodicity .....	51
Table 6-4: Tasks and their Priorities in PB Scheduling Algorithm.....	52
Table 6-5: Tasks and their Priorities in RR Scheduling Algorithm. ....	53
Table 6-6: Tasks and their Priorities in PBRR Scheduling Algorithm. ....	53
Table 6-7: Components and the Connected Pins to the MC. ....	54
Table 6-8: JTAG Connection with MC.....	56
Table 7-1: Execution Time for Tasks with Different Scheduling Algorithms .....	66
Table 7-2: Execution Time of Different Scheduling Algorithms by using Timer. ....	68
Table 7-3: Memory Consumption With or Without RTOS and with Different Scheduling Algorithms.....	68

## List of Abbreviations

GPOS	General Purpose Operating System
RTOS	Real Time Operating System
RT	Real Time
FSM	Full State Machine
TCB	Task Control Block
PC	Program Counter
PB	Priority Based
RR	Round Robin
FCFS	First Come First Serve
EDF	Earliest Deadline First
FP	Fixed Priority
SJF	Shortest Job First
PBRR	Priority Based Round Robin
OVF	Overflow
MCUs	Microcontroller Unit
LPG	Liquefied Petroleum Gas
$CO_2$	Carbon dioxide
PPM	Parts Per Million
LCD	Liquid Crystal Display
HAL	Hardware Abstract layer
MCAL	Micro Controller Abstract layer

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

GPIO	General Purpose Input Output pins
PWM	Pulse Width Modulation
ADC	Analog to Digital Converter
JTAG	Joint Test Action Group

## Chapter 1: Introduction

Operating systems consists of two types: firstly General-Purpose Operating system (GPOS) such as: Windows, Ubuntu, Android, and iPhone OS (IOS). And secondly the Real Time Operating System (RTOS). The time relevance and multitasking are the two main ways that GPOS and RTOS differ from each other.

GPOS is designed to be for personal use and GPOS is easy so everyone can use it easily in his daily life tasks. GPOS must be designed to support multi-tasks. In GPOS tasks are performed without priority bases so GPOS is allowed to execute a lower priority task before a higher one [1].

RTOS is a software layer found between application layer, and low-level drivers. RTOS Establish a framework that provides predictable behavior and guaranteed response times. RTS are used in: Anti-lock braking mechanisms, Online phone calls, Airbags, systems used in defense, such as RADAR and used in any system that offers up to the minute stock price information.

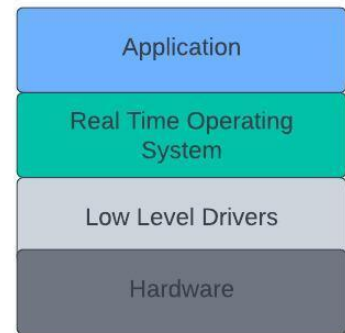


Figure 1-1: Embedded Systems Layers.

Table 1-1: Main Differences between RTOS and GPOS [2].

	RTOS	GPOS
<b>Application</b>	Used in Embedded applications	Used in PCs (Personal computers), phones, or any general-purpose device
<b>Latency</b>	Have their worst latency defined	Latency is not a concern of GPOS
<b>Task Scheduling</b>	Scheduling is time based	Scheduling is process based
<b>Priority inversion</b>	Have a mechanism to prevent priority inversion	No such mechanism is present
<b>Pre-emptive Kernel</b>	All kernel operations are preemptable	Not necessary

## 1.1. RTOS Architectures

Kernel RTOS are divided into two main types: monolithic kernel and microkernel according to their architectures [1, 2].

Table 1-2: Different Architectures of RTOS. [2]

Kernel RTOS's architectures	Monolithic kernel	Micro kernel
<b>Advantages</b>	<ul style="list-style-type: none"> <li>Runs the entire OS as a single binary file, which is quicker.</li> <li>In the same address space, a single big process handles both process scheduling and memory management.</li> </ul>	<ul style="list-style-type: none"> <li>Simple in debugging</li> <li>Lowering the amount of code running in kernel space lowers attack surfaces and increases security.</li> </ul>
<b>Disadvantages</b>	<ul style="list-style-type: none"> <li>Any error in the service can cause the OS to crash.</li> <li>Modifying and recompiling the OS is necessary for adding or removing services.</li> <li>Difficult to debug</li> </ul>	<ul style="list-style-type: none"> <li>Requires more context switching</li> </ul>

## 1.2. Several Types of Software's for RTOS

Table 1-3 shows the types of RTOS software, developer company and the availability "require license or not".

Table 1-3: Several Types of RTOS's Software's

Name	Availability	Company "Developer"
Free RTOS	Free "Open Source"	Amazon
Micruim os	Free "Open Source"	Silicon Labs
VxWorks	Needs license	Wind River Systems, Siemens EDA
RTAI	Free "Open Source"	RTAI Team
QNX Neutrino	Free "Open Source"	QNX Software Systems International Corporation "BlackBerry"
Azure RTOS	Not free requires license	Microsoft

### 1.3. Problem Statement

When designing a real time application meeting a time requirement “deadline” is essential to be taken into consideration as missing the deadline will affect the application’s performance or even lead to a failure in the application’s performance and may lead to a lot of problems or even a disaster that may affect a lot of people’s life and puts their life on a stake. This is why the developer must design and implement a well-designed application that ensure not to miss any deadlines. Also, choosing the best scheduling algorithm that is suitable for this specific application is a bit of a challenging task regarding the execution time, Which task should be executed first and based on what as an example: EDF, FP, RR or other scheduling algorithms and the CPU load. The annual incidence of LPG-related burns was 10% of all burning injuries for the first four years (2011-2014), but it increased to 26.94% in just the fifth year. LPG-related burns were primarily caused by gas leaks (81.03%), followed by improper operation (7.69%) and carelessness in the kitchen (2.05%).

### 1.4. Research Objectives

The aim of this research is to design and implement a real time embedded system’s application by using two different scheduling algorithms FP and RR and compare which is better in performance to detect any dangerous leakage gases such as LPG and  $CO_2$  or any abnormal in temperature in the kitchen then show it on screen and take an appropriate reaction which is by opening the ventilating fan in a certain direction or stop the ventilating fan.

This objective can be achieved by following this steps:

- Search for optimum sensors.
- Writing the code without RTOS.
- Implementing Priority based first algorithm on FreeRTOS.
- Implementing Round Robin algorithm on FreeRTOS.
- Change the time slice of the RR algorithm
- Implementing Priority Based Round Robin algorithm on FreeRTOS.
- Test the four codes on Proteus software.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

- Implement the hardware.
- Test the hardware by using Atmel ice debugger.
- Calculate the memory usage, time execution.

### 1.5. Research Questions

This research answers a lot of important questions like:

- What is RTOS?
- What are the main types of RTOS?
- What are the most famous RTOS?
- What is the task state?
- What is the task control block?
- What is a scheduling algorithm?
- What is the difference between preemptive and non-preemptive?
- What are the types of schedulers?
- What is the context switching?
- What is starvation, semaphore, deadlock?
- What is the difference between reentrant / non-reentrant functions?
- What is the interrupt latency?

### 1.6. Research Methodology

The procedures used in this research is first the author collect data about the topic by reading a lot of articles and papers to understand the topic, remark the faced problems and gaps then after identifying the problem the author try to provide some solutions.

In implementation phase the author write a code on Atmel Microchip Studio without RTOS then implement the Porteous schematic to verify that the application works as supposed to then resume writing the rest of codes then test it on Porteous again to ensure it works as supposed to, then burn the code through Atmel ICE debugger on the hardware implementation and test the results to ensure that everything works fine and extract the performance analysis.

## Chapter 2: Background

RTOS is considered a time critical system. Many industrial systems use RTOS. The most important achievement for the real time applications is to reach the action within the specified period before the deadline. There are 3 main types of RTOS: hard, soft, and firm [3].

### 2.1. Types of RTOS

First is the hard RT system: the operation must be executed within the specific time, not before or after the deadline. If the operation was not executed this means, there is a fatal error in the design. Examples on the hard RT system is the open of the airbags due to an accident any delays would put the user in higher risks, emergency braking system any delay in using the brakes may leads to a crash, an accident, and another example the missile defense system [3].

Second is the Soft RT system: it is not time dependent. If any delay happens, the operation will not be much affected, but the system benefit may decrease with time. Examples of the Soft RT system are personal computers, live streams, games, and web browsing.

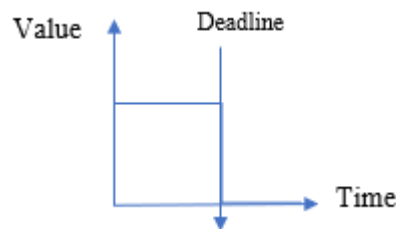


Figure 2-2: Hard RT system.

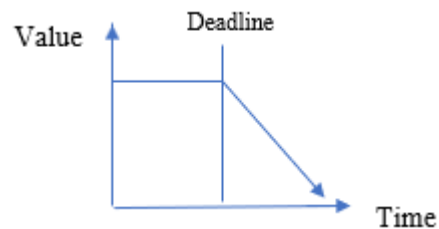


Figure 2-1: Soft RT system.

Third is the firm RT system: considered the same as soft RT system such as financial forecast systems.

The most used type of RTOS is the hard RT systems that are designed to execute and finish the application before the deadline. RTOS is used in many embedded applications. Hard RT systems are always considered a difficult challenge for the designers as the designers must ensure that the task is finished with certain time constraints. RTOS changes between the tasks according to the priority [3].



# Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 2-1: Different types of RTOS.

Types	Hard	Soft	Firm
<b>Main differences</b>	Finishing the task before or after the deadline is the same as if already missed the deadline.	The system can operate with a certain delay, but the performance may be affected and decreased	With few missed deadlines can be acceptable but more than that it will lead to system failure
<b>examples</b>	Missile defense system.	live streams and Games	Navigation control for weed killing robot.
<b>Effect of missing the deadline</b>	Missing target	Degraded in performance	Damage of crops

To ensure that the task is finished before the deadline, RTOS is dependent on both the completed time and the logical conclusion. As the delayed answer in hard RTOS is the same as the wrong one but in the soft it is acceptable [3].

## 2.2. Components of RTOS

1. The scheduler: this part handles the sequence of the tasks according to their priority.
2. Symmetric Multiprocessor: doing a lot of tasks that RTOS manages in parallel as if it was a multiprocessor.
3. Function Library: is a main part of RTOS as the function library is a layer between the Kernel RTOS and the code of the applications.
4. Memory Management: For each program save the memory.
5. Fast dispatch latency: the difference between the executed task and the beginning of executing another task from the ready state list.
6. User defined data objects and classes: used in c language.

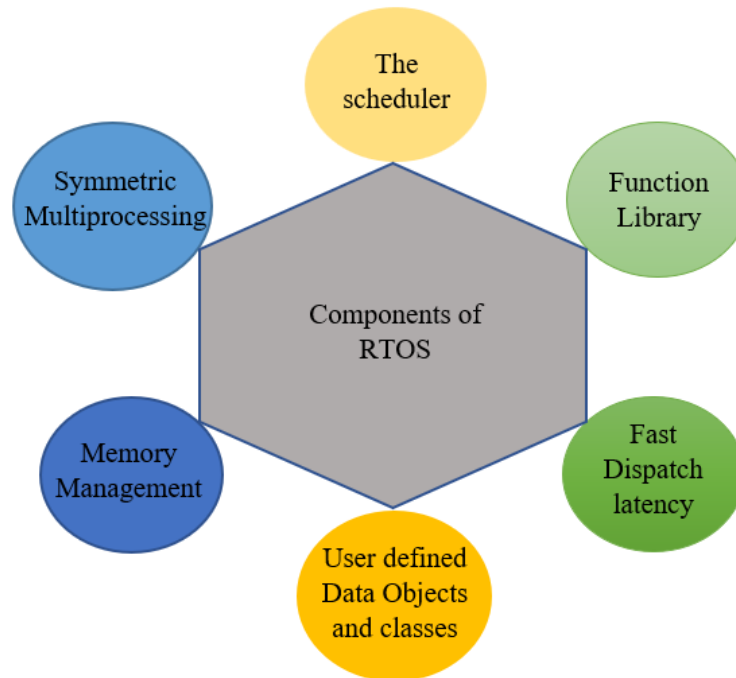


Figure 2-3: Components of RTOS.

### 2.3. Characteristics of RTOS

These are some features that RTOS are famous for:

- Reliability & stability: the OS will not malfunction or crash [3]
- Predictability: The application can be predicted as correct functionality with correct timing means good real time system [4]
- Performance: Speedily enough to meet the timing requirements
- Compactness: due to memory constraints in embedded systems applications RTOS is effective and compact.
- Scalability: This refers to the operating system's capacity to improve performance if new features are added [3].
- Availability: The OS has the chance to handles the requests and not to crash when OS the request is called [3]
- Usability/Portability: RTOS must be usable on different CPU architectures “microprocessors” and on different applications. [5, 3]
- Security: is the safeguard that keeps the system safe from any unwanted access from both inside and outside the system. RTOS should follow POSIX security

standards to support existing embedded applications. This function is effectively performed in different software such as VxWorks [3, 5, 4].

## 2.4. Advantages of RTOS

For big projects involving complex codes, RTOS is the best software to manage tasks. By using RTOS the project leaders can divide the software into small tasks that the other programmers can handle [6].

Synchronization and safety are improved remarkably by using RTOS. In small projects, the developers used to solve the Synchronization problem and to easily be used by other modules by using a global variable as the many usages of global variables may lead to many errors and safety problems especially in interrupt [6]. The global variables are so vulnerable to manipulation because they are shared by other functions or modules [6]. Semaphores may be a good solution for this problem [7].

The bugs get harder to find as the code expands so finding these bugs becomes more challenging. As a result, even for such tiny systems, development time may increase. This problem can be solved easily by using RTOS as the tasks can synchronize with each other without any corruption [6].

RTOS offers an abstraction layer that allows programmers to freely organize their code, produce cleaner code, and to quickly port their applications across many hardware platforms with minimal code adjustments [6].

In embedded systems applications, there are constraints on the hardware cost and constraints on the time of the development. Since the software designer must know certain peripherals (such as: timers, interrupts, etc...), also how to integrate them with the application code, implementing timing-related functionality in a small system without an RTOS can be complex. Any update, such as a longer delay time, would need a review of the code and peripherals by the developer to make the necessary adjustments. By implementing RTOS, the development time to tackle these issues is significantly decreased. Software developers can implement task delay, or timer handling processing using time management capabilities without having to be aware of the underlying hardware methods [6].

## 2.5. Disadvantages of RTOS

There are some disadvantages for using RTOS such as the difficulty and complexity in designing that increase the cost as the designer should focus to make the system response within the period and not be interrupted with some tasks lower in priorities. Also, RTOS consumes some of the memory both RAM and ROM by taking the files of the codes and include these files to the application, and RTOS does not support multitasking with round-robin scheduling [5, 6].

Task synchronization and other RTOS services must have a known execution time. The system designer can choose the API suite and structure the entire system based on these timing considerations and by utilizing the appropriate RTOS services. Therefore, designers need to be aware of the performance measures and benchmarking metrics used by RTOS [6].

## Chapter 3: Literature Review

When it comes to the speed and the accuracy of reaching a deadline in performing a certain application RTOS is very important to be used. Already a lot of researchers searched and published papers regarding RTOS so, the next paragraphs summarize various papers about RTOS from 2023 to 2019.

The literature review is divided into 3 main parts: RTOS application, Scheduling algorithms updates, and RTOS developments.

### 3.1. RTOS Applications

In [8] Xing et al. implemented an application for the protection of field workers in polluted sites. The application is a multi-component toxic gas monitoring system based on the STM32 microprocessor, and  $\mu$ COS-II embedded operating system. On the polluted site that is being restored, the system can track the concentration of dangerous gases such hydrogen cyanide, hydrogen Sulphide, carbon monoxide, and Sulphur dioxide in real time and show it on HMI. The sound light alarm will be activated when the poisonous gas concentration reaches the threshold level to alert the appropriate employees to evacuate quickly.

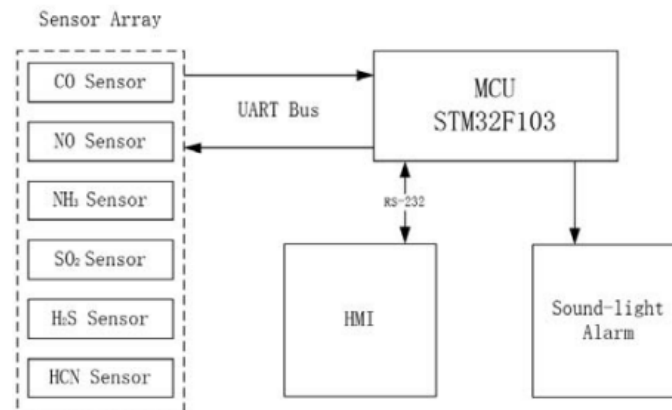


Figure 3-1: System Block Diagram [8].

In [9] Chelaru and Braescu implemented a real-time embedded system for maintaining and monitoring an aquarium in this research. A primary dsPIC33 microcontroller and a NodeMCU module that gives the system with IOT style features. Through the serial interface, the dsPIC33 talks to the NodeMCU module. Based on FreeRTOS, the dsPIC

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

application includes its unique task control, communication, and task synchronization features. The system provides features including auto fish feeding, fertilization, and aquarium lighting. It also gives water temperature information and regulates pH levels by adding CO<sub>2</sub>. Additionally, the system uses a DS1307 RT clock to access date and time data. The NodeMCU module runs a web server that displays aquarium data.

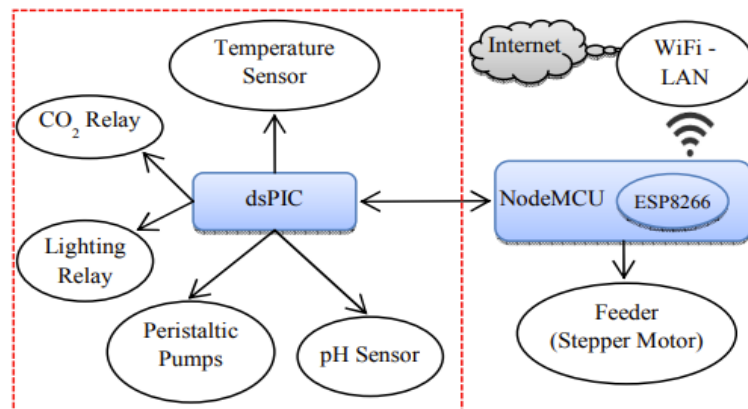


Figure 3-2: System Block Diagram [9]

In [10] Palatty, Chandra and Sivraj set up a system for rendering and capturing videos. to analyze RTOS functionality and design restrictions in real-time systems that must follow time limits. The system either presents the gathered video feed on the LCD display or broadcasts the encoded video feed by creating a UDP connection with the remote host, depending on the software configuration. The ATSAMV71 platform serves as the base for all application functions, which are all implemented as FreeRTOS tasks. The OV7740 image sensor and an LCD display are examples of hardware elements that assist the system with its various operations. The system profile acquired using SystemView, a trace visualization tool, is used to estimate performance. The study provides a thorough understanding of the tasks that are required, the required computation, and the time limitations that are present in the system.

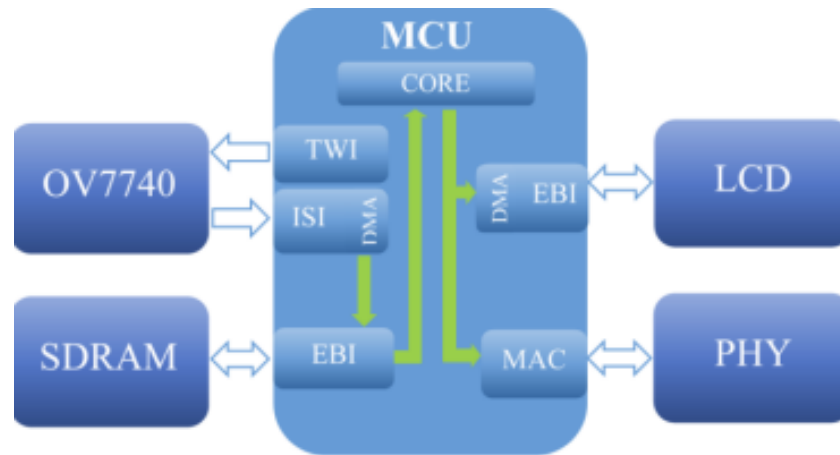


Figure 3-3: System Block Diagram [10].

In [11] Arm et al shows a potential issue and examines measurement approaches for determining performance and accuracy. The benefits and drawbacks of different measuring techniques are examined. Moreover, factors defining the functionality and characteristics of RTOSs operating on multi-core systems are defined. Deep descriptions of task period jitter and measuring semaphore taking are included. The ESP32 development kit's XTENSA dual-core processor running FreeRTOS is used to measure the operations. As a result, the unexpected rescheduling to another core extends the time it takes for FreeRTOS operations to execute. In the future, the authors intend to use advanced sensing methodologies (profiling, RTOS function tracing) to guarantee more precise outcomes and validate the measurements.

In [12] Gunawan et al. implemented a real time chicken farm on an Arduino. Different sensors will be used to measure the gases, temperature, and humidity levels, exhaust DC fans and heat lamp will be used to control to the ideal level. The results of the experiments demonstrated that RTOS on Arduino exceeds non-RTOS on Arduino in terms of performance. This is because RTOS allows for the assigning of task priorities, allowing for the execution of higher priority tasks first. Future work will involve remote monitoring, remotely logging data, and remotely controlling the Arduino system by connecting it to the Internet.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

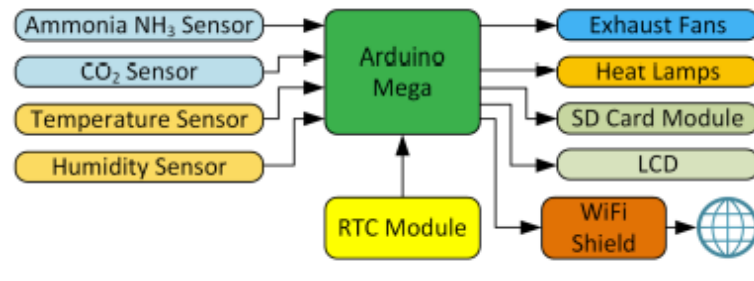


Figure 3-4: Block Diagram of the Chicken Poultry Farm Application [12].

In [13] Septian, Misbahuddin and Arkan implemented in this research, an air quality monitoring system that measures variables including CO<sub>2</sub>, temperature, humidity, and heat index is constructed using an Arduino Nano and Internet of Things technologies. Display and transmit activities, among others, are run simultaneously using FreeRTOS. CO<sub>2</sub>, temperature, humidity, and heat index are all sensed by MQ135 and DHT22, respectively. ESP8266 Wi-Fi module will be used to periodically send data to a web server using the secure HTTPS POST protocol. A web server is used to establish a secure online application and to receive sensor parameters on the back-end side of the system, allowing users to monitor it remotely.

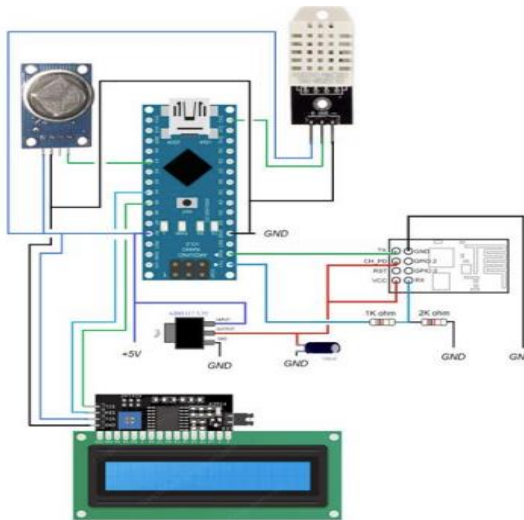


Figure 3-5: Block Diagram of Air Quality Monitoring System [13].

In [14] Sarma and Reddy implemented a device that the proposed device might make use of technological tools that is totally focused on an integrated board and its improved database for real-time recording of all sensor modifications. The microcontroller which is used on this board is atmega328. A warning signal would be sent to your smart phone if the



## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

car's speed exceeded the limits, indicating that all sensors were operating simultaneously. The Bluetooth module of the owner's cell phone is connected to the microcontroller.

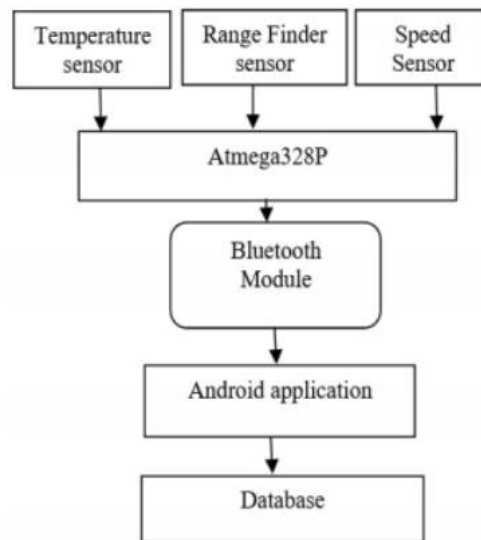


Figure 3-6: Block Diagram of the Application [14].

In [15] Ungurean and Gaitan compare the task context switching times for 4 different RTOSs used on ARM Cortex TM-M based microcontrollers: FreeRTOS,  $\mu$ C-OS/II, RT-Thread, and Keil RTX. When using events, semaphores, or mailboxes for synchronization, these RTOSs monitor the time required for task context switching. The tests are integrated on ARM CortexTM-M4/M0+ based MCUs. The most popular RTOS is FreeRTOS, but this type has the longest context switching time compared to all other types. The most effective results were achieved with Keil RTX.

In [16] Juhász, Pletl and Molnar introduces a new RTOS kernel along with some of its capabilities, such as tasks, scheduling, thread-safe communication, context switches, etc. The response time and memory footprint of this RTOS are faster than those of several comparable market-leading ones found in ARM Cortex-M (FreeRTOS, EmbOS, KeilRTX). The test is conducted using an ARM Cortex-M4-based STM32F429I development board. Moreover, the code is compiled using the ARM Compiler V5 in O3 + speed optimization in the Vision IDE (Keil).

In [17] Juhász, Pletl and Molnar introduces a new RTOS kernel along with some of its capabilities, such as tasks scheduling, context switches, etc. The response time and memory

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

footprint of this RTOS are faster than those of several comparable market-leading ones found in microcontrollers (FreeRTOS, EmbOS, KeilRTX). The test is conducted using an ARM Cortex-M4-based STM32F429I development board. Moreover, the code is compiled using the ARM Compiler V5 in O3+ speed optimization in the Vision IDE (Keil).

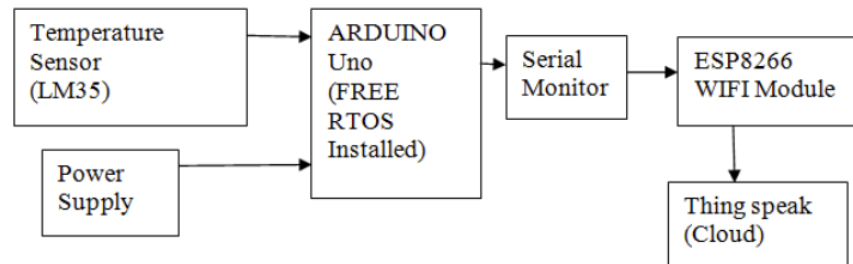


Figure 3-7: Block Diagram of Remotely Monitoring System [17].

In [18] Khomenko and Khomenko explain a few issues with the creation and testing of FreeRTOS-based applications. To make this process more practical, using the software application Tracealyzer is recommended. Benefits of this usage have been demonstrated for common problems encountered throughout the development and debug phases. The hardware used in all cases is Nucleo-F429 board (build on STM32F429 microcontroller). Additionally, all data traces were captured in snapshot mode. The advantages of this software tool have been demonstrated through situations where it has helped in resolving issues with task periodicity and Deadlock states between two tasks using shared resources and mutexes.

In [19] Akhoury, et al says that the Real-Time Operating System (RTOS), Micruim OS-III, is used to schedule tasks on two of the microcontrollers that allows the system to be sensitive to the priority and time limits of each task. and by using Segger System View and the Sampled Graph feature in IAR, a thorough comprehensive study of the application of the RTOS has been given beside a strong quantitative analysis. The third microcontroller is run and controlled purely through interrupts from the other two CPUs, in contrast to this OS-based design. The research illustrates the usage of a partial OS-based and partial interrupt-based task switching model and summarizes its benefits and drawbacks. The research also explains the many steps of onboard processing of thermal camera images, including image compression and data encoding algorithms before to transmission that serve

to reduce data loss during transmission and enable error detection and repair upon payload data arrival.

In [20] Edmaier, Ueter and Chen found that there is a serious problem which is a task might begin by running on a processor then moving on to a GPU. This problem is much more complicated in the case of RT constraints which is the execution within strictly specified time boundaries. If the scheduling algorithms are not well designed, real-time restrictions may result in significant resources under use. There is a solution to this issue which is Self-suspension and segment-level fixed-priority scheduling algorithm. This method divides tasks into self-suspension and calculation portions that are alternated sequentially. If a process tries to access hardware that is already being used by another task, it may self-suspend. In this research, the authors suggest and examine various FreeRTOS implementations of the segmented self-suspension task model. Moreover, the authors assess the overhead of several implementations on the NXP M40007 IoT-module.

In [18] Khomenko and Khomenko explain a few issues with the creation and testing of FreeRTOS-based applications. To make this process more practical, using the software application Tracealyzer is recommended. Benefits of this usage have been demonstrated for common problems encountered throughout the development and debug phases. The hardware used in all cases is Nucleo-F429 board (build on STM32F429 microcontroller). Additionally, all data traces were captured in snapshot mode. The advantages of this software tool have been demonstrated through situations where it has helped in resolving issues with task periodicity and Deadlock states between two tasks using shared resources and mutexes.

In [21] Akgun and Gohringer claims that the task scheduling service must be called on a regular basis, which adds to software overhead and eventually causes jitter. However, by using reconfigurable systems, the associated overhead can be reduced or even removed. Furthermore, deadlines might not be reached since dynamically modifying power dissipation of reconfigurable systems changes how quickly applications run. Therefore, a thorough analysis of how these optimizations would affect real-time capabilities is required. Which is presented in this paper and focus on offloading RTOS components while taking power consumption on reconfigurable platforms into consideration. So, the task scheduling

of FreeRTOS has been transferred to a co-processor to scale voltage and frequency on the XC7Z020. This resulted in a 38.9% decrease in the task scheduling's execution time.

### 3.2. Scheduling Algorithm Updates.

In [3] Ismael et al. reviewed scheduling algorithms for RTOS, summarize and classifies other papers and compares their different features. The authors talk about concept of scheduling algorithms (Soft, firm, and hard deadlines), components of RTOS (scheduler, fast dispatched latency, memory management, user defined data objects and classes, and function library), properties of RTOS (reliability, stability, scalability, availability, usability, and security) and scheduling criteria (CPU usage, performance and waiting time). The authors talk about different scheduling algorithms and their advantages such as: LST, SJF, FIFO, RR, EDF and FP.

In [22] Teraiya and Shah implemented two dynamic scheduling algorithms for the soft RTOS (EDF and LST) as well as two static scheduling strategies (RM and SJF). Results from the testing of algorithms using a recurring work set are gathered. In a similar setup, the authors had tracked the success rate (SR) and effective CPU usage (ECU) for all methods. EDF and LST, which are dynamic algorithms, have been discovered to work well under load but poorly in overload, while RM and SJF, which are static algorithms, have also been found to fail to schedule a particular task underload but perform well in overload. Practical research has been conducted on a large dataset. Datasets consist of more than 7000 process sets, each process set has 1 to 9 processes. It has been tested 500 times to confirm its accuracy in all circumstances.

In [23] Nascimento and Lima proposed forward a framework for multiprocessor RTS that allows for the scheduling of both hard and soft jobs with flexible deadlines by using one of two EDF configurations. Two specialized servers that are in charge of delivering temporal isolation and slack reclamation are the foundation of their approach. For the soft and hard scheduling algorithms, they presented EDF with Local Queue and EDF with Global Queue scheduling strategies. The combination of these methods improves the soft task reaction time average. Extensive simulation is used to analyze the results, which show that it is possible to effectively use the processing capability that is available,

significantly lower than average response time of aperiodic jobs, maintain hard deadlines, and maintain a low soft deadline miss ratio.

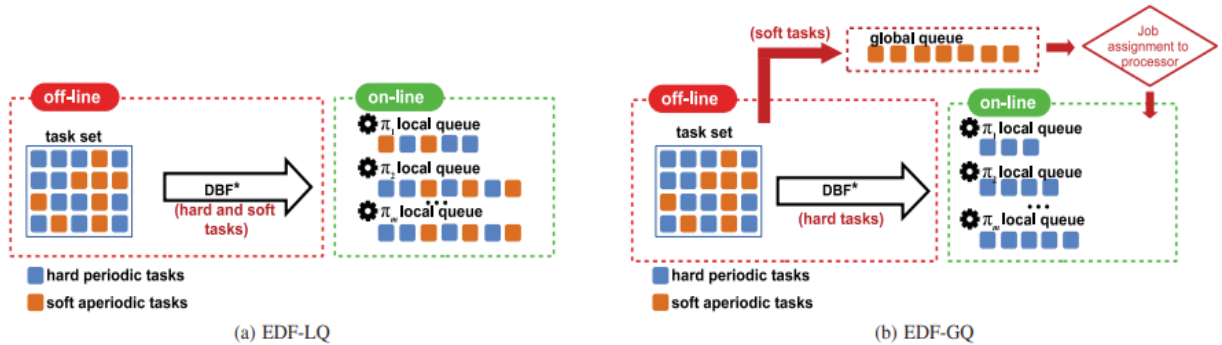


Figure 3-8: Two Possible Queuing Configurations for Tasks [23].

In [24] Putra discussed in this research the priority preemptive scheduling algorithm. With a priority scale of 0 to 10, with 0 being the lowest importance and 10 being the highest. This research discusses two case studies. The result of these two studies is that the average waiting time for priority processes varies. For case 1, the mean turnaround time and the mean waiting time was more than that in case 2. Case 1 is completed more time than in case 2.

In [25] Sinha et al. suggest a new scheduling algorithm called ESRR (Efficient Shortest Remaining Time Round Robin), which combines two preemptive scheduling algorithms called RR and SRTF. In comparison to RR, ESRR decreases total waiting time and turnaround time, as well as waiting times for shorter processes. It also enables longer processes to execute more quickly than SRTF.

In [26] Abu-Dalbouh created a brand-new CPU scheduling method called the mix PI-RR algorithm. The suggested approach decides which jobs should execute and which should wait to use a mixing of RR and PB scheduling techniques. This innovative technique solves the drawbacks of both scheduling algorithms. The proposed mixed PI-RR algorithm produced better CPU scheduling results, according to the performance metrics and

overcoming some issues with the other two algorithms. Also, Zouaoui, Boussaid and Mtibaa in [27] implemented the same approach for scheduling algorithm.

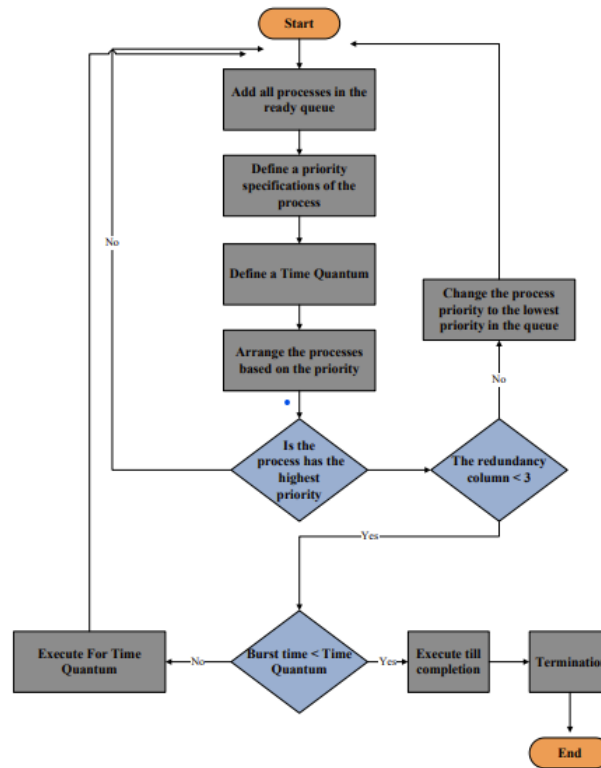


Figure 3-9: Flowchart of the Proposed PI-RR Scheduling Algorithms [26].

### 3.3. RTOS Developments.

In [28] Unguritu and Nichițelea says this research analyses the characteristics of AUTOSAR (Automotive open system architecture) multicore by an adequate RTOS which coordinating the cores and their tasks in accordance with various strategies. Then the research provides a new and expanded OS with adaptive characteristics that enable each core to manage not only its own activities but also the essential tasks of other cores, but only when necessary. This will enhance the safety perspective of automotive embedded systems. The simulation results show the expected system behavior when taking the suggested task migration technique into consideration. The future work may be tested by using a real multicore system or an emulator.

In [29] Lencioni, Loubach and Saotome proposed a general methodology to help embedded systems designers choose and select a certain RTOS to meet design limitations, like static memory allocation of kernel objects and a layout with a small memory footprint.

The process is applied to a reduced example that serves as an illustration, and the outcomes are examined. Future studies could evaluate the suggested methodology against more targets constrains, like commitment to POSIX standards and adjustments to real-time scheduling methods. POSIX is the abbreviation for Portable Operating System Interface which is a series of standards to keep operating systems compatible with one another.

In [30] Begum et al. claim that an ineffective distribution of resources could result in a system deadlock. Based on the least number of resources that are accessible and the processes' highest resource demands, this study suggests a new mechanism for safe state detection in a system. Their approach collects each process resource requirement into a linked list, which makes it easy to assess whether a request exceeds the resources that are easily accessible. In their experiments, they evaluated their plan with a few different models, including the first banker's algorithm. The findings showed that their suggested technique offers less computational complexity and less space complexity when compared to the alternative solutions. In the future, they intend to use the new algorithm that has been provided to avoid deadlocks and to carry out more deep research to test their approach against some undesirable processes.

In [31] Xu, Zhang and Ge found that modelling and analyzing the time requirements for a certain application in the beginning are very important to avoid redesigning or recoding the system at later stage which will cost more money and waste of time. Also, the existing work is not enough to support the modelling for both microkernels based RTES and the difference configurable policy. Their main contribution consists of two parts: First, proposing domain specific language (DSL) for the timing analysis modelling of the microkernel based RTES followed by defining and implementing DSL as unified modelling language (UML) profile. Second, proposing static timing analysis for RTES represented by DSL, where a timing analysis tree and consistent execution rules are developed to analyze the variability in a broad view. To demonstrate how to apply their framework to a real-world robot controller system, they used the scheduling policy as an example in the case study. In the future work, they are enhancing their strategy to support periodic tasks as well as the multi-core processors.

In [32] Raffeck, Ulbrich and Preikschat say that in their previous research They were able to successfully solve the drawbacks of full migration in RTS by lowering the associated cost and improving its predictability. They were able to find useful migration points by using static analysis, and then they generated static schedules that migrated tasks at these points. In this research they extend this strategy to dynamic scheduling by giving an OS knowledge about advantageous migration points so OS can decide on migration at runtime.

In [33] Mamone, et al. says that for safety-critical embedded systems, such in the automotive, and aerospace. reliability is now one of the major problems, between the hardware layer and the software/application layer of an embedded system, there is typically an intermediate layer made up of middleware and the operating system (OS). The reliability analysis considered the application layer only most of the time as the middle layer execution time is so small that the probability of a fault affecting the system is less than application layer. However, a hardware problem might be anticipated to have potentially hard consequences every time it propagates to the middle-layer as an error, especially to the OS. This research paper aims to investigate the Single Event Upset (SEU) or known as single event error, is a fault that affect Real-Time Operating System (RTOS) reliability. The approach focuses on the FreeRTOS data structures and variables that are most important for analysis when using a software-based fault injection which is a software testing technique made for improving software performance. The result of many fault injection experiments allowed them to analyze which data structures are more sensitive to SEUs, facilitating future advancements such as selective OS hardening based on FI data.

In [34] Bosio, Redaudengo and Savino examined the reliability of FreeRTOS under the influence of Single Event Upset faults. To target the most important variables and data structures, the methodology is based on fault injection. Results show that the OS fault tolerance is selective, opening the door to more specifically designed fault-tolerant techniques such selective OS hardening.

In [35] Serino and Cheng studies the current and the future state of RTOS for CPS (Cyber Physical System) time sensitive which is a computer system which monitor or controls the mechanism by a computer-based algorithm. This research shows the benefits and the drawbacks for RTOS and GPOS in CPS applications. Moreover, scheduling, kernel, and



priority inversion have been compared across a few RTOS, including VxWorks, RTLinux, and FreeRTOS. They examined several WCET (Worst-Case Execution Time) estimation tools. This research also discusses future developments in RTOS, such as multicore RTOS, new RTOS architecture, and RTOS security.

In [36] Alagalla and Rajapaksha found that researchers have investigated a wide variety of strategies for synchronization. Still, there isn't enough research being done to determine the best method for improving the precision and effectiveness of real-time operating systems. In addition, research analytically explains the various synchronization platforms that are used in the computing industry, such as hardware-oriented synchronization, on-chip memory handling, location-based network systems configuration, and backup synchronization. This research specifically focuses on the issues and difficulties with the current RTOS and Distributed RTOS (DRTOS) Systems. This paper explains the best practices for developing distributed node networks using multi-core processors. By applying different methods, including lock-based, lock-free, semaphore-based, mutex-based, and hardware-related high accuracy requirements, it will suggest several techniques to increase efficiency.

## Chapter 4: RTOS Operation

RTOS takes the tasks to be executed according to scheduling algorithm which identifies the tasks' state. Each task has its own task control block as explained in part 4.2.

### 4.1. Task States

- **Running:** The task that is now running is referred to as being in the running state and This task has the highest priority.
- **Ready:** The task is ready to run, but it cannot as another task with an has the same priority or higher is already executing.
- **Blocked:** The task is in the blocked state whenever it waits for a certain event to happen or completing a delay period or the unavailability of a resource. The blocked tasks are not available for scheduling until the event happen.
- **Suspended:** The suspended task is found in a lot of real time software such as FreeRTOS. Like tasks that are blocked, suspended tasks cannot be chosen to move into the running stage, however suspended tasks do not have a timeout. Instead, when a specific command is written, tasks only then enter or leave the Suspended state.

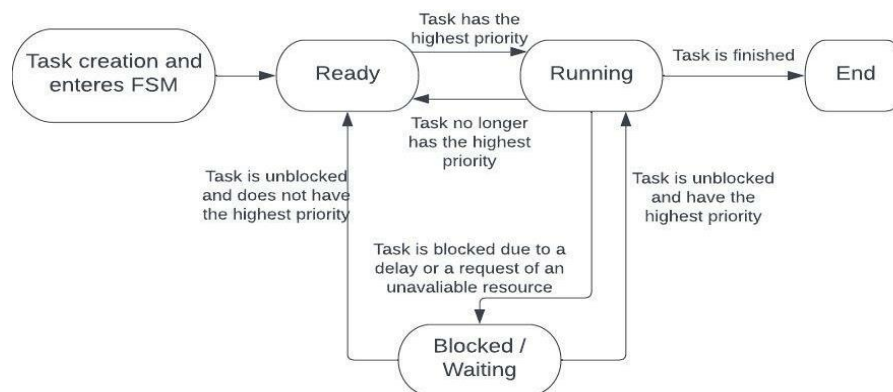


Figure 4-1: Full Task State Machine.

As a result of starvation, an issue with priority scheduling systems, a process that is prepared for CPU tasks may have to wait an endless amount of time for its start since it has a low priority. A steady stream of higher-priority tasks may prevent a

low-priority task from ever receiving the CPU consideration in a computer system that is overloaded [26].

## 4.2. Task Control Block

When a task is created automatically assigned to TCB. TCB is a data structure to maintain the state of the task and stores the task in the RAM [37].

Figure 4-2 contains Task ID which is a number to identify the task. Priority which contains the task priority with respect to other tasks. Status which is the task state. Registers to save any data. PC program counter to save the context switching which is the process of memorizing the address of the latest executed task before being interrupted by another task. Stack Bottom pointer is pointer that contains the address of the bottom of the Stack. Stack size identify the size available for this task. Context switching counter is simply a counter for each context switching [37].



Figure 4-2: Task Control Block.

## 4.3. Scheduling Algorithms

Scheduling is the process through which the process management chooses another task based on a certain algorithm after removing the active task from the CPU [38, 24, 39, 27]. The tasks were defined based on the deadline, completion time, estimated time of arrival, and resource requirements. Based on their properties, scheduling algorithms are divided into two different classifications: static and dynamic [40].

Scheduling algorithms are activated by events, which are typically brought on by hardware or software interrupts (system calls). A fatal error occurs in the system if the scheduling method is not correctly implemented [41].

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 4-1 compares preemptive, non-preemptive with respect to the definition, interrupt, waiting-response time, examples, and task diagram to clarify the process.

Table 4-1: Preemptive and Non-Preemptive Scheduling [38, 24, 39, 27].

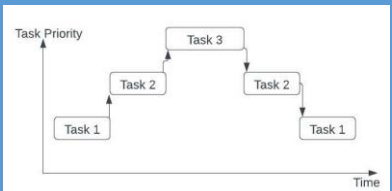

Parameter	Preemptive Scheduling	Non-Preemptive Scheduling
<b>Define</b>	When a task switches from the running state to the ready state or from the waiting state to the ready state, preemptive scheduling is used.	When a task ends or changes from the running state to the waiting state, non-preemptive scheduling is used.
<b>Interrupt</b>	Tasks can be interrupted in between.	Tasks cannot be interrupted until it terminates itself or its time is up.
<b>Waiting and Response time</b>	Waiting and response time of preemptive Scheduling is less.	Waiting and response time of the non-preemptive Scheduling method is higher.
<b>Examples</b>	Round Robin, Shortest Remaining Time First (SRTF), etc.	First come First Serve and Shortest Job First (SJF), etc.
<b>Scheduling task diagram</b>		

Table 4-2 shows the differences between static and dynamic scheduling, definition, speed, and complexity.

Table 4-2: Static and Dynamic Scheduling.

Parameters	Static Scheduling	Dynamic Scheduling
<b>Define</b>	Static priority scheduling is a kind of scheduling method where all jobs have the same fixed priority	Dynamic priority scheduling is a kind of scheduling method where the priorities are determined while the system is being used.
<b>Speed</b>	Not as fast as Dynamic	Faster than Static
<b>Complexity</b>	Less complex	Complex Hardware

Figure 4-3 shows the main types of scheduling algorithms and their types as shown each scheduling algorithm is divided into 2 categories preemptive and non-preemptive and each part of them is divided into 2 parts static or dynamic and then provides an example for each type.

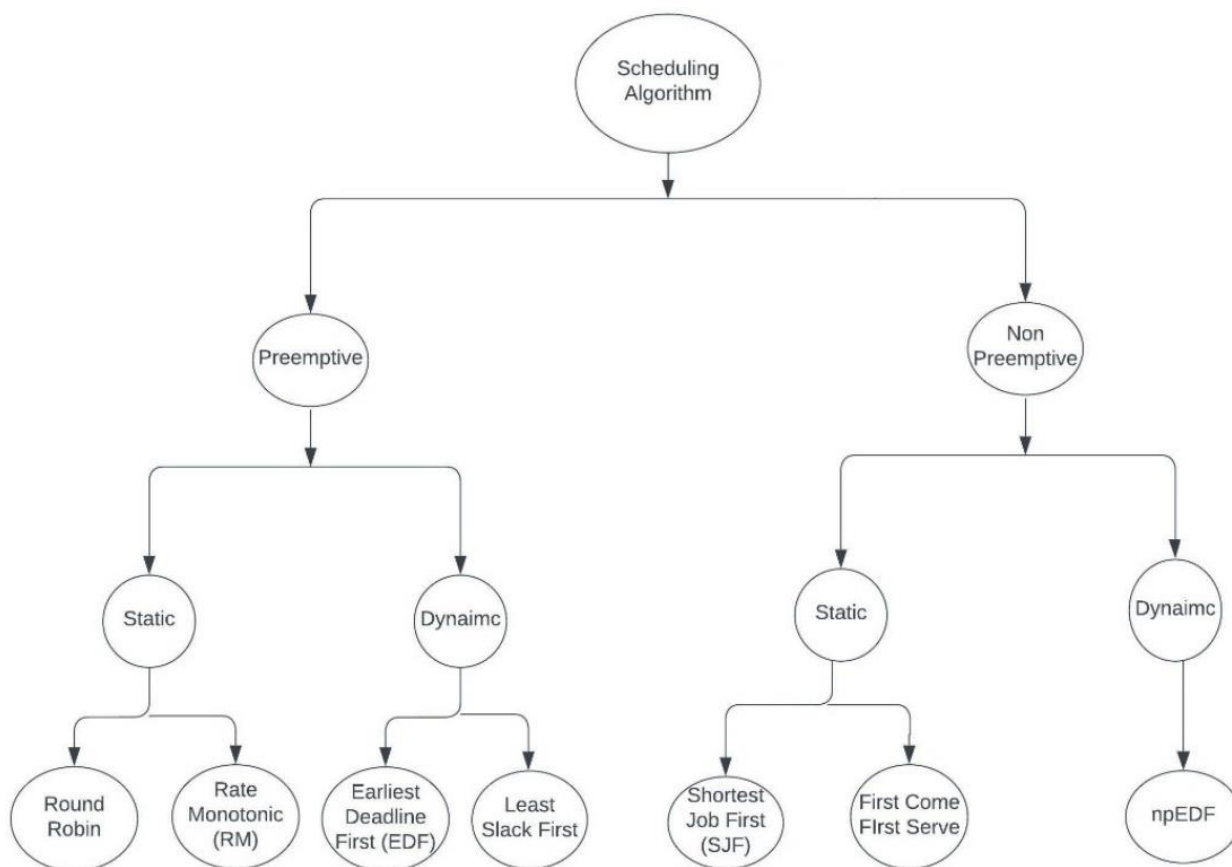


Figure 4-3: Scheduling Algorithm Classifications.

#### 4.3.1. Shortest Remaining Job First

SRJF is the preemptive version of SJF. The task with the smallest remaining time achieves the highest priority [25, 42].

#### 4.3.2. Round Robin (RR)

A circular queue implements the RR scheduling method. The selected task is the first in the queue, and the insert is completed at the end of the queue. Tasks do not have a priority system, each task executes for a certain time, as shown in figure 4-4, when the interrupt occurs, the running task is added before last task of the queue when the time finishes, and the first task in the queue is activated so the execution time is reduced [41, 43, 42].

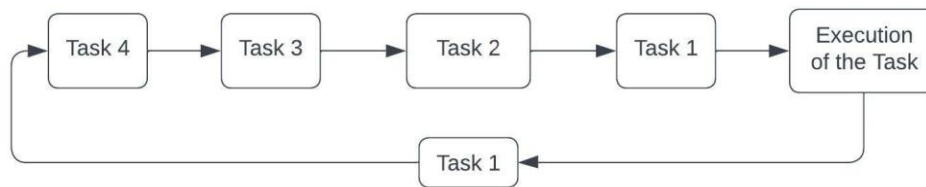


Figure 4-4: RR Tasks in the Queue

#### 4.3.3. First Come First Serve (FCFS)

One of the famous non preemptive scheduling algorithms is FCFS. FCFS or The FIFO algorithm gives priority to the queue that is ready and entered first for CPU execution. Therefore, the time for process waiting must be dependent on the time that the process was really executed in the past. [43, 42]

#### 4.3.4. Shortest Job First (SJF)

The SJF is considered a Static priority non preemptive scheduling algorithm. The task with the smallest execution time achieves the highest priority [40, 42].

Figure 4-5 illustrates the SJF algorithm's flowchart. The scheduling algorithm will operate and determine the execution time for each task whenever a new task is added or the task that is currently being executed is ended. The new task chosen is the one that has the least execution time [40].

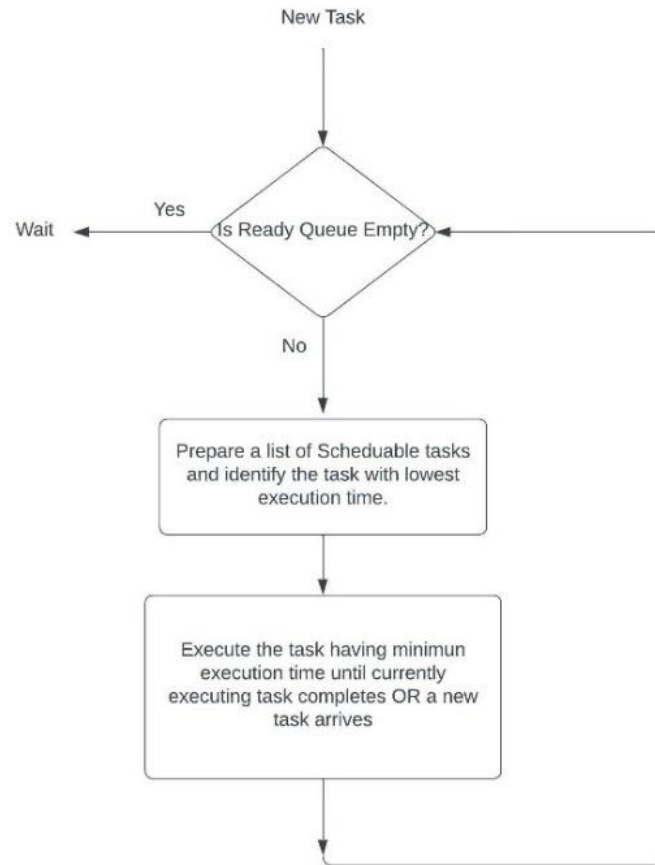


Figure 4-5: The SJF Algorithm's Flowchart

#### 4.3.5. Deadline Monotonic (DM)

Deadline Monotonic is a static preemptive priority scheduling. DM is a static priority scheduler that prioritizes tasks based on their fixed deadlines. The instance of the ready task in the queue with the shortest deadline is the first one to be executed by DM at any given time and have the highest priority since, the priority is inversely proportion with deadlines. DM is a preemptive scheduling algorithm, which means that if a task with a larger priority is provided, the running task will be interrupted, and the CPU will be assigned to the larger priority task. And if more than two tasks have the same due date, DM will choose one at random to complete first [44].

#### 4.3.6. Rate Monotonic (RM)

Monotonic Rate is considered a preemptive scheduling algorithm that gives static priority to the tasks. Each task's deadline falls inside its period, the computation time is fixed, and the switching time is recognized as null. Each task's period determines its priority, the

smaller the period, the bigger priority. A running task loses the prioritized when a higher priority job enters the CPU because the algorithm is preemptive [41].

#### 4.3.7. Earliest Deadline First (EDF)

The EDF is considered a preemptive dynamic scheduling algorithm and considered the most practical algorithm. Highest priority is given to the task with the shortest or nearest deadline. Dynamically assigned and modified priorities are used. EDF is quite effective compared to any scheduling algorithms [45, 41, 44].

Figure 4-6 illustrates the EDF algorithm's flowchart. The scheduling algorithm will operate and determine the nearest deadline for each task. whenever a new task is added or the task that is currently being executed is ended. The new task chosen is the one that has the nearest deadline [45].

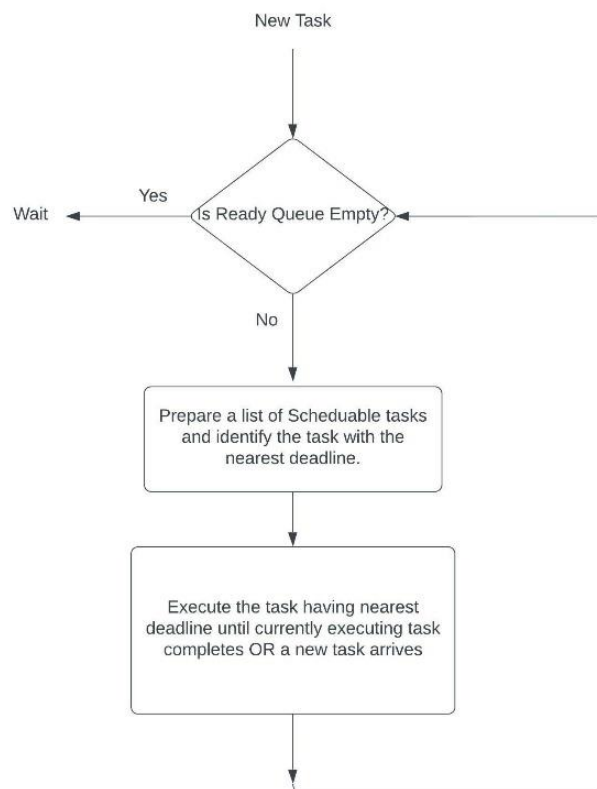


Figure 4-6: The Earliest Deadline First (EDF) Algorithm's Flowchart.



#### 4.3.8. Least Slack Time First (LST)

The LST is considered a dynamic preemptive scheduling algorithm. The task with the smallest slack time achieves the highest priority. The following equation gives the definition of the slack time [40].

$$\text{Slack time} = \text{Deadline} - \text{Remaining execution time} - \text{Current time} \quad (1)$$

Figure 4-7 illustrates the LST algorithm's flowchart. The scheduling algorithm will operate and determine the slack time for each task based on Equation 1 whenever a new task is added or the task that is currently being executed is ended. The new task chosen is the one that has the least amount of slack time [40].

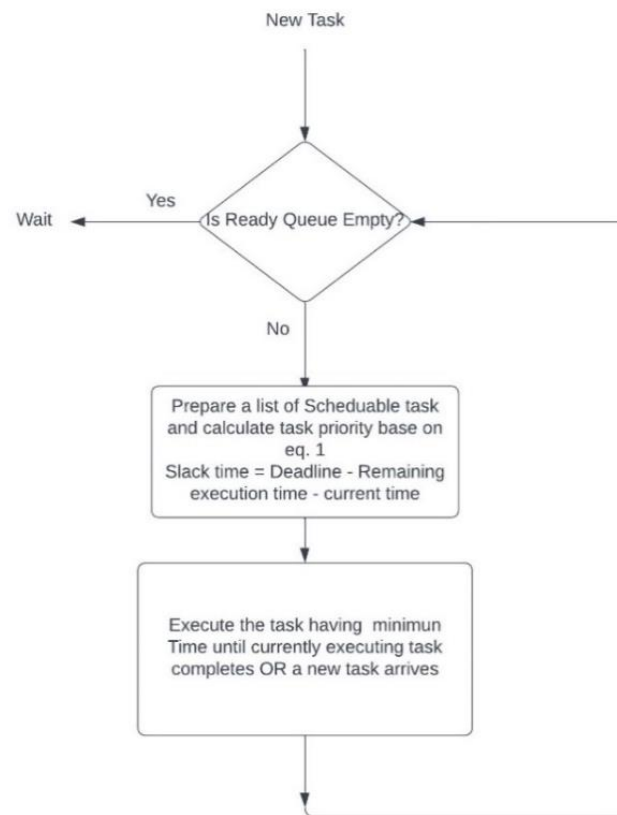


Figure 4-7: The LST Algorithm's Flowchart.

#### 4.4. Semaphore

In RTOS, a semaphore is a variable that affects how many different processes can access a shared resource or a task to synchronize their activity. There are 2 main types of semaphores: binary, counting semaphores [7]. Binary semaphore has only 2 values 0

and 1 (locked/unlocked, available/unavailable). Counting semaphore depends on 8,16 or 32 bits so from 0 to 255, 65535 or  $2^{32}$ .

Re-entrant function can be interrupted at any time and resumed later without damaging the data. This function can be used by more than one task, but the shared data may be corrupted so semaphores is used to save the value from corruption.

#### 4.5. Deadlock

Deadlocks, sometimes known as deadly embraces, occur when two tasks are each unknowingly waiting for a resource “Mutex” that the other task is holding. As shown in figure 2-11, while holding Mutex A, Task 1 wants Mutex B. Task 1 cannot continue until it has both the Mutex A and Mutex B. While holding Mutex B, Task 2 wants Mutex A. Once Task 2 obtains both Mutex A and Mutex B, it can proceed. To avoid deadlock, there is two methods: acquiring all the resources before proceeding or acquiring the resources in the same order and release the resources in reverse order [30].

Mutex is a global or shared value most likely be a binary one that can be accessed by other tasks. So, if a task took the mutex the task can read or edit the mutex value.

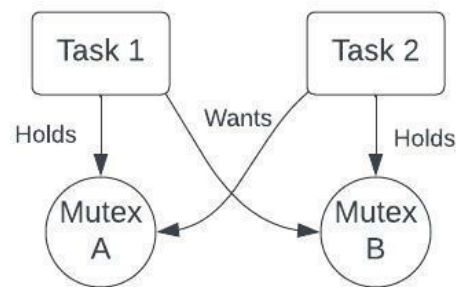


Figure 4-8: Deadlock Explanation Graph.

## Chapter 5: Proposed Performance Measurement on Different Scheduling Algorithms

There are two types of measuring the performance of the RT application. By using ready software or by implementing codes using timer or Ticks. Both methods needs a hardware implementation and a debugger connected to the application's hardware.

### 5.1. Criteria of the Performance that May be Evaluated

- Task execution time: The total time taken by the task to be fully performed.
- Context switching: The time between saving the address of the current task and going to another task which is mostly higher priority which happened as a result of using interrupt or the task needs a mutex “variable” from another task.
- Total execution time: The total execution time of the application.
- Power loss: The power loss through implementing the hardware
- Total memory consumption: The Memory size
- Interrupt latency: The time between the interrupt request and the and the beginning of execution of the ISR.
- CPU usage: The percentage of load have been applied to the CPU.
- Stack usage and heap memory.
- Worst Case Execution Time (WCET)

### 5.2. Software to Measure the Performance

These are some of the famous Software to measure the performance of RT applications.

#### 5.2.1. SEGGER System View

System View is software for the visual inspection of any embedded system. System View offers far more than a debugger does, by providing comprehensive visibility into a program and a detailed understanding of what occurs during runtime. This is especially helpful for creating and managing complicated structures with numerous activities and instances [46].

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

A small software module on the target side that contains SystemView and RTT is required. The data is gathered and formatted by the SystemView software before being sent to RTT. In specific circumstances, such as interrupt start and interrupt termination, the target system invokes SystemView methods to track events. In the RTT target buffer, SystemView records these tasks along with a programmable, highly accurate timing. Timestamp accuracy can reach 1 CPU cycle, or 5 ns on a 200 MHz processor [46].

### 5.2.1.1. Continuous Recording

Any device that supports J-Link RTT technology "J-link debugger probe" may capture continuously in real time. When a program is running, RTT needs to be able to read memory via the debug interface. This specifically includes all Renesas RX devices and the ARM Cortex-M0, M0+, M1, M3, M4, and M7 processors. While the application is being executed, System View can constantly record target execution in real time. This makes it possible to analyze system behavior over a longer time frame without having to use a big target memory buffer. RTT and the SYSVIEW modules take up less than 2 Kbytes of total ROM space. Around 600 bytes of RAM are sufficient for most applications [46].

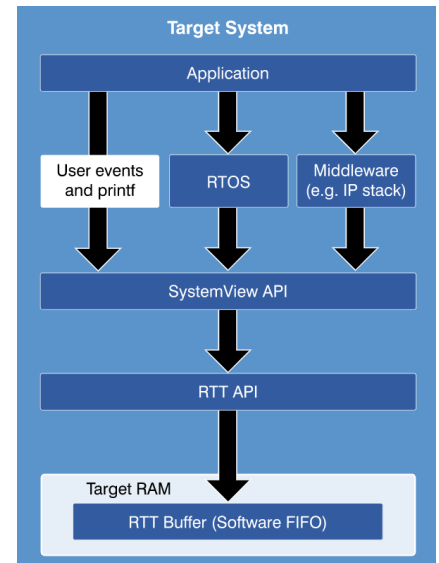


Figure 5-1: SEGGER SystemView Working Flow Diagram.

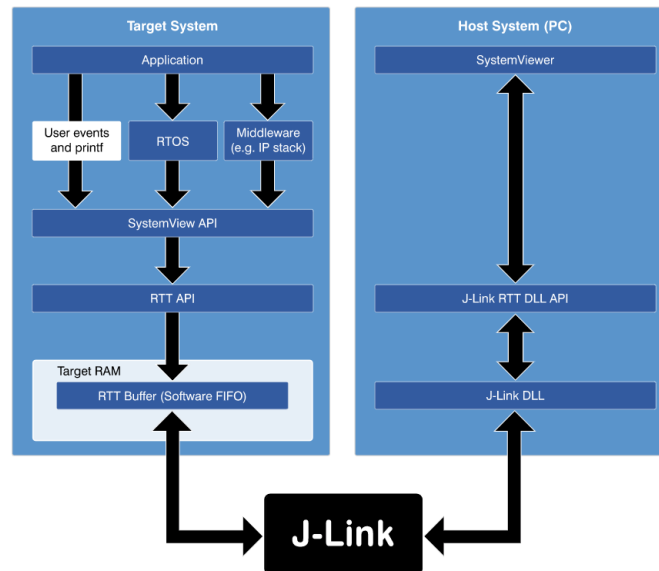


Figure 5-2: Continuous Recording Flow Diagram [46].

#### 5.2.1.2. Single Shot Recording

SystemView is able to keep recording data into its buffer until it is full even if the target device does not support RTT or when no J-Link is being used. so JTAG debugger probe can be used. For a respectable period of time, single-shot capture provides insight into the startup process or event-triggered behavior in any application [46].

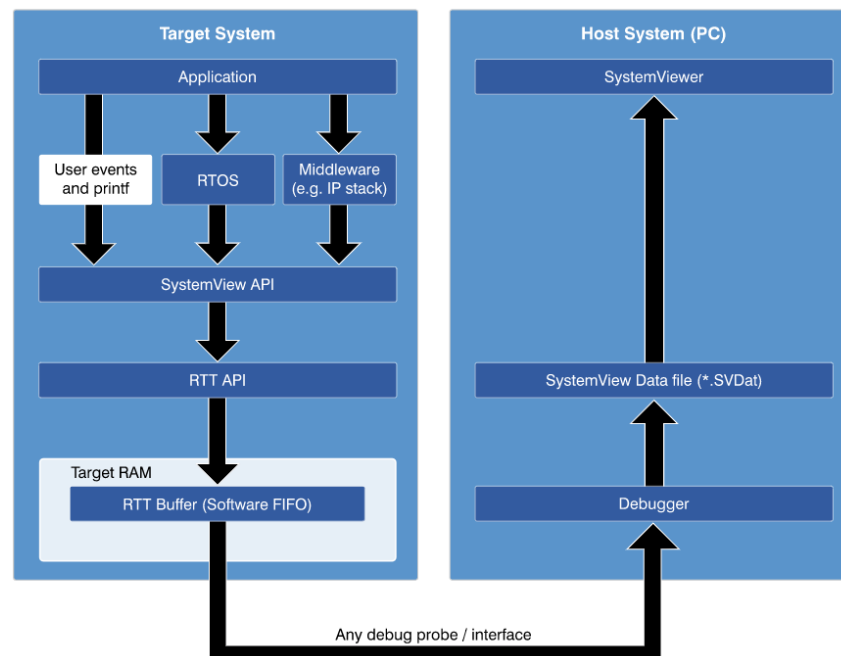


Figure 5-3: Single Shot Recording Flow Diagram [46].

### 5.2.2. Percepio Tracealyzer

Software event traces have been recorded by Tracealyzer using highly effective software instrumentation. This can either be retained in the target RAM until needed or transmitted to the host application views. Numerous CPU models are supported by Tracealyzer, including ESP32 and Arm-based devices. by adding a few lines of code Tracealyzer supports new CPUs. Leading embedded software platforms and tools, such as FreeRTOS, VxWorks, Micrium OS-III, Segger J-Link, and Eclipse/GDB are all supported through integrations. This software needs license which the company provides a free licenses for 10 days [47].

### 5.2.3. SimSo Simulator

SimSo is a software simulator that shows RT multiprocessor design that employs statistical models to account for the effects of caches as well as various scheduling overheads (scheduling decisions, context changes). It enables quick simulations and quick development of scheduling policies using Python and a Discrete-Event Simulator (SimPy). This simulator have many scheduler such as: FP, RM and EDF [48].

### 5.1.4. Keil MDK by ARM

Keil MDK supports ARM microprocessor architecture only(M-family). All Cortex-M devices are supported by MDK-Core, which is built on the Vision IDE. Assembler, linker, and highly optimized run-time libraries are all included in the Arm Compiler for Embedded, which is designed specifically for Arm Cortex-M type processors only for the best efficiency and code size. In MDK-Professional, a functional safety qualified version is supplied [49]. CMSIS is an abbreviation of The Common Microcontroller Software Interface Standard.

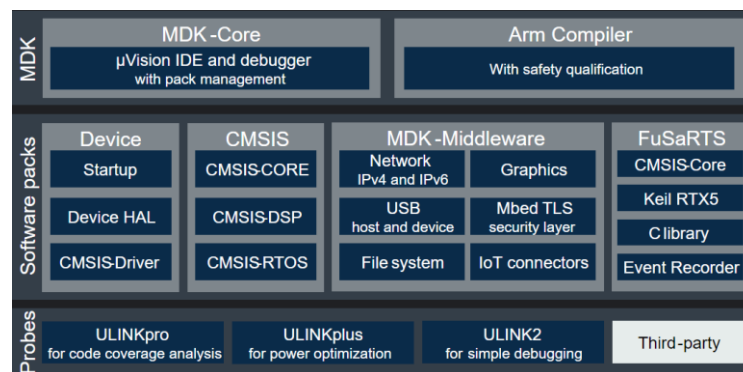


Figure 5-4: Layering of Keil MDK [49].

### 5.3. Analytical Method to Measure the Performance

There are two main types to get the performance. The first one is by using built in APIs for the RTOS and the second one is by using a the built in timers in the microprocessor.

#### 5.3.1. Atmel ICE

All the next analysis would not be applicable without using Atmel ICE the debugger to debug and trace the global variables of the code and see the working principle if there is any problem in implementing or designing the code.



Figure 5-5: Atmel ICE.

Atmel ICE debugger debugs both AVR and SAM “ARM” microcontrollers by using many connections such as: JTAG and aWire / PDI-wire / debugWIRE interfaces as this interfaces based on the MCUs bits number: 8, 32 bit. Figure (5-6) show the connection of the debugger with Adapter board using AVR port.

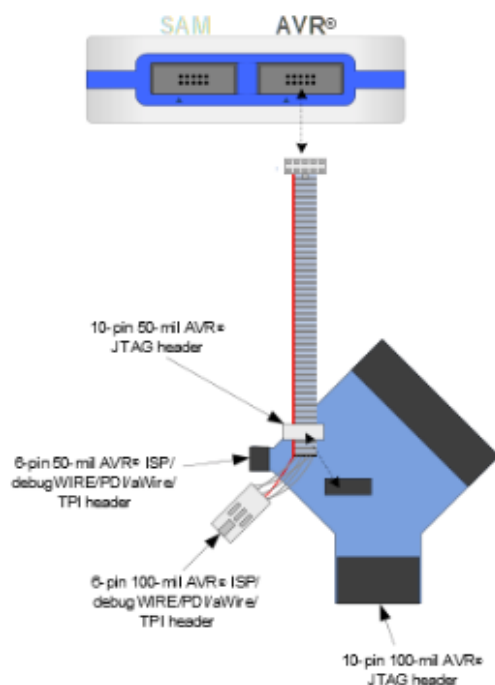


Figure 5-6: ATMEL ICE Board Connection with Adapter Board

### 5.3.2. By using the RTOS APIs

As an example, In FreeRTOS there is an API called “xTaskGetTickCount”. The user can know the number of ticks per each task and from these ticks the user can know the exact time taken to finish this task when this API is called the following procedures shown in the next figure diagram begin to execute. First the developer must initialize a global variables at the start of the task and at the end of the task then after finishing executing the task. The developer can calculate the executed task time by subtracting the global variable that is called end with that another one that called start.

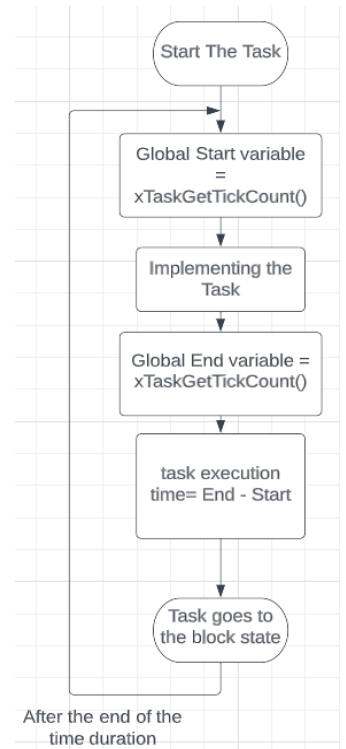


Figure 5-7: Flow Diagram of the APIs Method to Get the Task Execution Time.

### 5.3.3. By using the Build in Timer

First implementing on timer 1 as timer 0 is already taken by RTOS ON to facilitate the operation such as: context switching and the interrupts. Then the author have to initialize the working of the timer which is in this case as the following steps:

- Put zeros on “TCCR1A” to make the timer works in overflow mode.
- Then the “TCCR1B” have to be XORed with the “CS10” bit to work with no prescaling and preserve the values of the other bits.
- Then the “TIMSK” register have to be XORed with the “TOIE1” bit to work/enables with the overflow interrupts and not to change any other bits in the register.
- Then put the start value of the counter “TCNT1” = 0 then begin to increase linearly until the counter reaches the OVF
- Then enables the SREG to interrupt. And waits for an interrupt when the counter reach overflow “0xFFFF = 65,536 ” to clear the counter value of the “TCNT1” then increase the global variable of completed overflows counter by one which indicates the number of overflows happened to this task or the whole program. Then set the flag “TOV1” to one to clear it.



- To get the timer value. First, we have to know the value of the timer when the timer reaches the overflow “equation (2)” then multiply this number with the global variable of completed overflows counter. Then add the multiplied number of  $T_{cycle}$  with the value of TCNT1 or as shown in equation (3).

$$T_{cycle} = \frac{1}{Frequency_{operating}} = \frac{1}{1 * 10^6} = 10 \mu sec$$

$$time\ wanted = no.\ of\ counter * T_{cycle} \quad (2)$$

$$T_{OVF} = time\ wanted\ to\ reached\ OVF = 65,536 * 10^{-6} = 0.065536 = 65.536\ msec$$

If the user receives the global counter = 2, and the timer counter = 32,768 .

$$time\ reached = global\ counter * T_{OVF} + timer\ counter * T_{cycle} \quad (3)$$

$$time\ reached = (2 * 65.536 * 10^{-3}) + (32,768 * 10^{-6}) = 0.16384\ sec$$

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

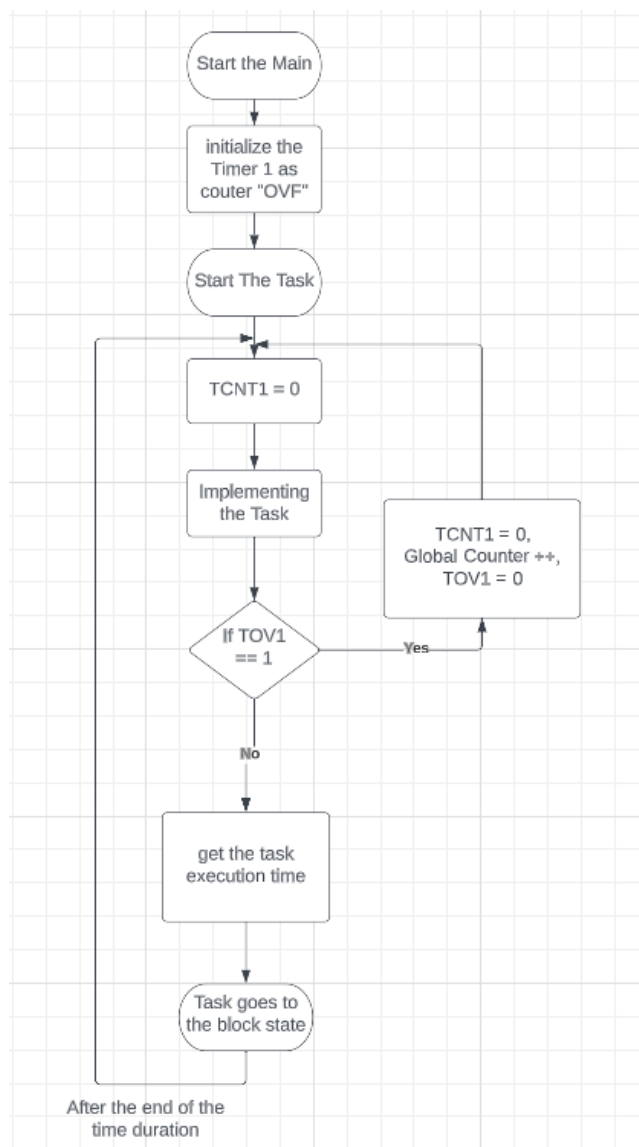


Figure 5-8: Flow Diagram of the Timers Method to Get the Task Execution Time.

## Chapter 6: Case Study Kitchen Air Control System.

First the author chose to implement three scheduling algorithms which is Priority Based, Round Robin and then Priority Based Round Robin Scheduling Algorithms also implementing the same application but without using RTOS and then compare between the two scheduling algorithms and without RTOS regarding the execution time and the memory size. These scheduling algorithms is widely used in FreeRTOS so the author decide to compare their execution time.

Table 6-1: Comparison Between PB and RR Scheduling Algorithms [50].

Scheduling Algorithm	Priority based	Round Robin	Priority Based Round Robin
Advantages	<ul style="list-style-type: none"><li>• Tasks can be given different priority levels according to their importance, urgency, or other factors.</li></ul>	<ul style="list-style-type: none"><li>• The CPU is divided equally across all running Tasks.</li><li>• There is no starvation because RR has a cyclical characteristic.</li></ul>	<ul style="list-style-type: none"><li>• Reducing the starvation</li><li>• The CPU is divided equally across all running Tasks.</li><li>• Tasks can be given different priority levels according to their importance</li></ul>
Disadvantages	<ul style="list-style-type: none"><li>• Lower-priority tasks may starve and be continuously delayed if high-priority tasks consume a lot of CPU time.</li></ul>	<ul style="list-style-type: none"><li>• Setting the time slice too long could result in a slow response to perform tasks, while setting it too short would add overhead</li></ul>	<ul style="list-style-type: none"><li>• Setting the time slice too long could result in a slow response to perform tasks, while setting it</li></ul>

	<ul style="list-style-type: none"> <li>Choosing which task receives which degree of priority</li> </ul>	<p>and reduce CPU efficiency.</p> <ul style="list-style-type: none"> <li>A very large time quantum causes RR to change to FCFS.</li> </ul>	<p>too short would add overhead</p>
--	---	--	-------------------------------------

## 6.1. The Components used and The Method

The application basically is kitchen air control system. The application contains 2 sensors MQ2 gas sensor and LM35 and take the appropriate decision which is opening the motor clockwise as the fan will work to decrease the temperature if the motor was anti clockwise the fan will work as ventilating fan and display both sensors' readings along with fan state on LCD 20\*4. Figure (6-1) shows the whole block diagram of the application. The application is implemented on ATmega32A the frequency used in this application is internal oscillator 1 MHz.

The author chose this application specially to help save people from dangerous situations. According to [51], The annual incidence of LPG-related burns was 10% of all burning injuries for the first four years (2011-2014), but it increased to 26.94% in just the fifth year. Patients between the ages of 21 and 60 were most frequently affected by LPG burns (73.85%). LPG-related burns were primarily caused by gas leaks (81.03%), followed by improper operation (7.69%) and carelessness in the kitchen (2.05%). so, this application will help to decrease these percentages

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

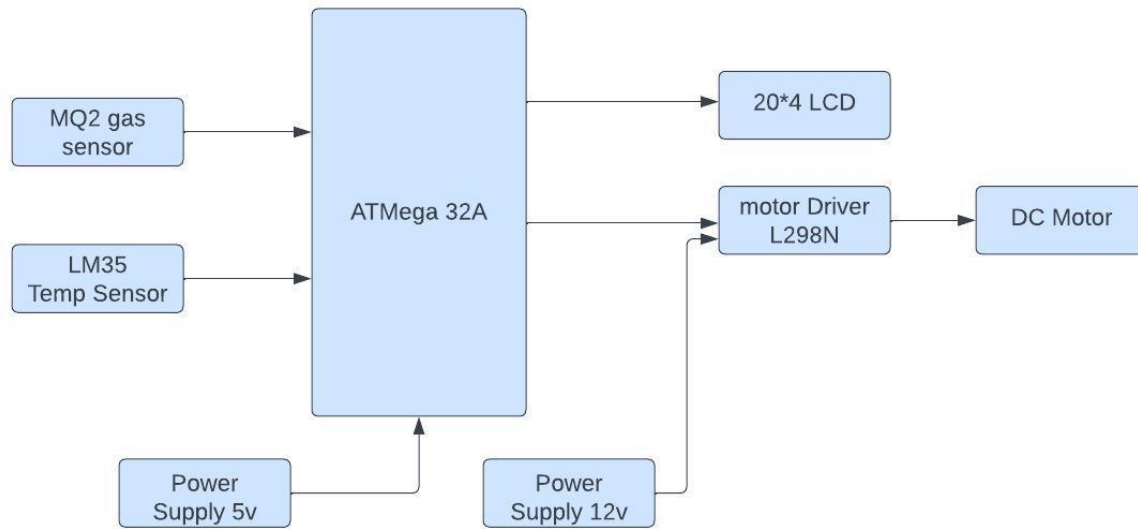


Figure 6-1: Block diagram of the Implemented Application.

Figure (6-2) illustrates the flow chart of the whole application. Where at the beginning the sensors have to be initialized and then begin to take their reading after that keep checking the condition of the temperature then if the temperature is higher than 30°C the fan should be on by making the motor rotates clockwise. Then goes to the next condition to make sure that the gas sensor is less than a critical value. If the LPG value is more than 2000 or the smoke value is more than 400 then the fan should be enabled by the movement of the motor anti

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

clockwise. After that the values should be printed on the lcd and then the loop will repeat continuously from the sensor reading part.

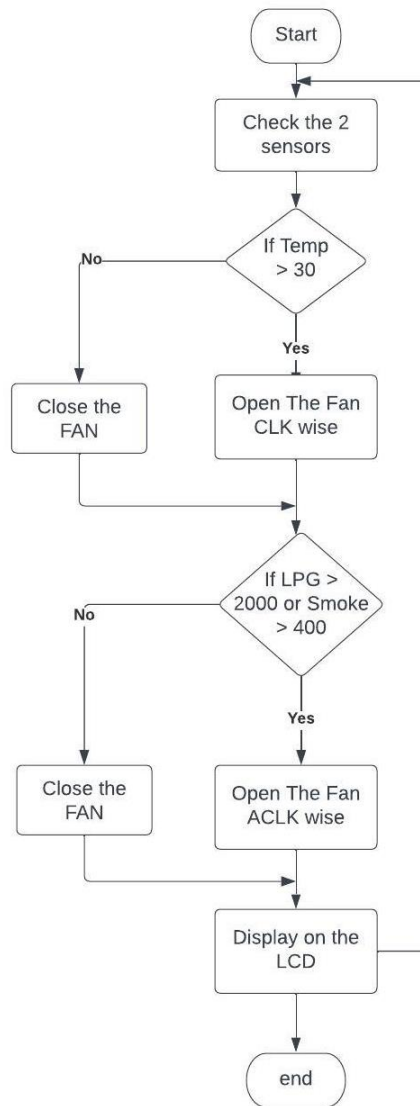


Figure 6-2 :Flow Chart of the Implemented Application.

### 6.1.1. The MicroController

ATMega32A is an 8-bit MC based on AVR that has a lot of features such as:

- 40 input/output pin with 32 KB Flash, 2 KB SRAM, and 1 KB EEPROM.
- A 32 mapped registers (R0-R31).

- ATmega32A has JTAG standard that enables programming and debugging the codes also contains normal communications protocols such as: SPI, I2C, and UART.
- ATmega32A has 3 Timers/counters “ Two 8-bit and one 16 bit ”
- 4 PWM channels
- 8 inputs pins for 10-bit ADC

#### 6.1.2. Temperature Sensor

LM35 is one of the used sensors in this application. This is a Temperature sensor that has a 3 pins one for the input voltage that varies from 4 v to 30 v , another pin for ground and the last pin Vout which is Temperature Sensed voltage “analog” output. Temperature sensing range from -55 °C to 150 °C with a linear scale factor 10 mv / °C. Equation number 4 shows the calibration which is how to calculate the value of the temperature from the Vout variations.

Where  $sensor\ max.\ temp = 150\ ^\circ C$ ,  $ADC\ Refrence\ voltage = 2.56\ v$ ,  
 $ADC\ maximum\ value = 1023$ , and  $sensor\ max\ volt\ value = 1.5\ v$

$$Temp = \frac{V_{out} * sensor\ max.\ temp * ADC\ Refrence\ voltage}{ADC\ maximum\ value * sensor\ max\ volt\ value} \quad (4)$$

#### 6.1.3. Gas Sensor

MQ2 gas sensor is used to senses LPG, smoke, Alcohol, Propane, Hydrogen, Methane and Carbon Monoxide. This sensors has four pins, one for input voltage 5 v, ground, digital output that gives 0 if there is a presence of abnormal gas and gives 1 if everything is normal, and an analog output that provides an exact value of ppm of each gas. PPM is abbreviation for parts per Million.

The MQ2 is a sensor powered by a heater. A two-layer "anti-explosion network" made of fine stainless-steel mesh is used to shield the sensor as a result. Because the designer thinks that the users will be monitoring combustible gases, the designers make sure that the heater element inside the sensor does not trigger an explosion. Additionally, an anti-explosion network shields the sensor from damage and prevents the particles from penetrating the chamber so that only gaseous components can pass through the chamber [52].

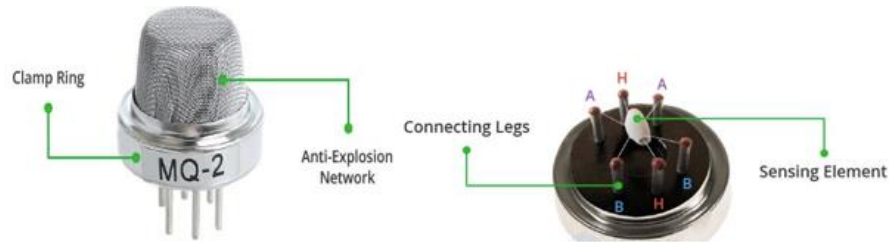


Figure 6-3: MQ2 Shape and Internal Structure [52].

A ceramic based on an aluminum oxide ( $Al_2O_3$ ) and tin dioxide ( $SnO_2$ ) core makes up the cylindrical sensing part. The most essential element is tin dioxide since it reacts with flammable gases easily. The sensor's region is constantly being heated to the proper working temperature thanks to the ceramic material, which also increases the effectiveness of heating. In conclusion, the Sensing System is made up of Platinum wires and a Tin Dioxide coating, despite the Heating Unit is made up of a Nickel-Chromium inductor and an aluminum oxide-based ceramic [52].

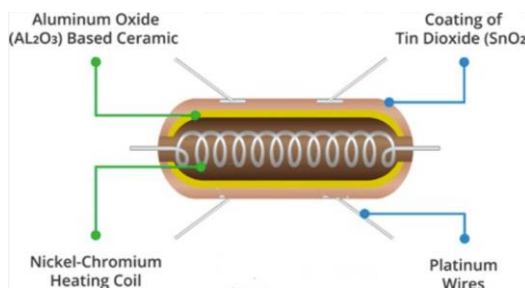


Figure 6-4: Sensing Element of MQ2 Sensor [52].

The author uses MQ2 to detect smoke and LPG which is liquified Petroleum Gas which may be propane, butane or both together. From the datasheet of this sensor, there is 2 curves the author is interested in LPG and Smoke curve. To get the value in ppm the author have to follow this rules. The author took an average of 15 reading to make the reading as accurate as possible. Also, by taking into consideration that  $R_o = 9.83$  from the curve below which is clean air factor also that the range of  $R_L$  from  $5k\ \Omega$  to  $47k\ \Omega$ , the author assumed that  $R_L = 10\ K\Omega$ .



First the LPG take any 2 points from the graph such as point 1 = ( 200 , 1.6 ), point 2 = ( 10000 , 0.26 ) , the author normalized both points to be within the same range. point1 = (  $\log(200)$  ,  $\log(1.6)$  ) = ( 2.3 , 0.20 ) , point 2 = (  $\log( 10000 )$  ,  $\log ( 0.26 )$  ) = ( 4 , -0.58 ) ,by substituting in equation 5,6 by “y” which is  $\frac{R_S}{R_o}$ , the user can get the value of LPG in ppm. LPG range from 200 ppm to 5000 ppm. The maximum value LPG can reach before being poisoned is 2000 ppm [53].

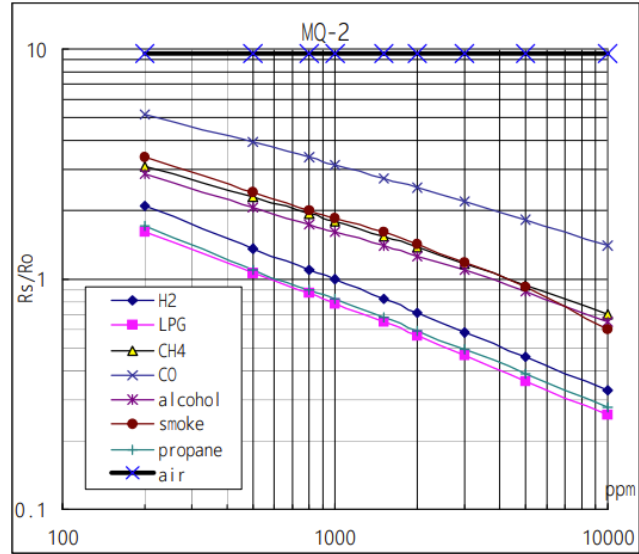


Figure 6-5: Sensitivity Characteristics of the MQ2.

$$m = \frac{(Y_2 - Y_1)}{(X_2 - X_1)}$$

$$m = \frac{(-0.58 - 0.2)}{(4 - 2.3)} = -0.46$$

$$X = \frac{1}{m}(Y - Y_1) + X_1$$

$$X = \frac{1}{-0.46}(\log(Y) - 0.2) + 2.3 \quad (5)$$

$$\text{Value in ppm} = 10^X \quad (6)$$

First the Smoke take any 2 points from the graph such as point 1 = ( 200 , 3.4 ), point 2 = ( 10000 , 0.62 ) , the author normalized both points to be within the same range. point1 = (  $\log( 200 )$  ,  $\log ( 3.4 )$  ) = ( 2.3 , 0.53), point 2 = (  $\log ( 10000 )$  ,  $\log( 0.63 )$  ) = ( 4,-0.2 ) , by substituting in equation 7,8 by “y” which is  $\frac{R_S}{R_o}$ , the user can get the value of smoke in ppm. The maximum value smoke can reach before being poisoned is around 400 ~ 800 ppm [54]. More information is shown in the following table [55, 56].

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 6-2: PPM Values of CO and the Time Before Critical Symptoms [55, 56, 54].

PPM value	Time and symptoms
<b>50 ppm</b>	Normal exposure for adults no more than 8 hours period
<b>200 ppm</b>	Headache, fatigue, nausea and dizziness after 2 to 3 hours
<b>400 ppm</b>	Frontal headaches within 1 hour. Life threatening after 3 hours
<b>800 ppm</b>	Dizziness and nausea within 45 min. unconscious within 2 hours. Death within 2 to 3 hours
<b>16,000 ppm</b>	Headache, Dizziness and nausea within 20 mins. Death within 1 hour

$$m = \frac{(Y_2 - Y_1)}{(X_2 - X_1)}$$

$$m = \frac{(-0.2 - 0.53)}{(4 - 2.3)} = -0.43$$

$$X = \frac{1}{m}(Y - Y_1) + X_1$$

$$X = \frac{1}{-0.43}(\log(Y) - 0.53) + 2.3 \quad (7)$$

$$\text{Value in ppm} = 10^X \quad (8)$$

### 6.1.4. LCD

LCD has many types. The one used in this project is 20 \* 4. LCD has 16 pins first pin ground then Voltage supply that is normally 5v then variable voltage to change the contrast of the brightness of the lcd then Rs for register select either sending data or commands then R/w which is read or write which is always reading so this pin should always be grounded then E which is the enable pin for the lcd. Then pin 7 to 14 to send commands or data. The last two pins for back light but normally the author connected the anode with the positive 5 v and the cathode with the ground to give more light to the lcd. The sending bits could be proceeded be 2 methods 4 bits or 8 bits depends on the number of the available pins in the micro controller

and the speed of sending the data or commands each instruction on one time in case of 8 bits or on two time in case of 4 bits.

#### 6.1.5. DC Motor

DC Motor consists of a stator and rotor and can work by energizing one of the wires of the coils so the motor will rotate to a specific direction if the wires are reversed the motor direction will be changed. The DC motors can be controlled by using H bridge motor drivers such as: L298, L298N and L293D.

#### 6.1.6. Motor Driver

L298N Motor Driver works with stepper and dc motors. This driver can work with two dc motor and can their control speed and direction. This driver have many input pins such as: 12v input for the driver, 5v input for switching logic gates, ground, IN1 & IN2 controls the direction of the motor, EN enables the PWM of the motor and OUT1, OUT2 for the motors. The speed of the motor can be controlled by PWM signal “Duty cycle” where the duty cycle is the time taken by the on part with respect to the total time. The direction of the motor is determined by the usage of h-bridge [57].

PWM can be calculated by using this equation:  $\text{duty cycle} = \frac{T_{on}}{T_{on}+T_{off}}$ , where we can calculate the frequency of the pulse by using this equation  $\frac{1}{T_{on}+T_{off}}$

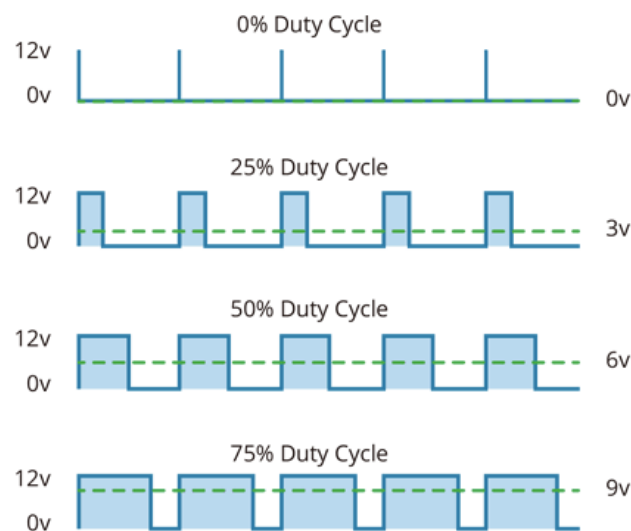


Figure 6-6: PWM Percentage Variations [57].

## 6.2. Layering Software Architecture

All the written codes goes under layering architecture design where there is a 3 major layers which is the application then the HAL which is Hardware Abstract Layer and MCAL which is MicroController Abstract Layer.

HAL where anything that their code won't be changed by changing the microcontroller such as: sensors, LCD and keypad. MCAL is anything that is related to the MCU's hardware such as: GPIO "General Purpose input output pins", ADC "Analog to digital converter" and Timer configurations. The RTOS layer is found between the application layer and the hardware abstract layer.



Figure 6-7: Layering Architecture.

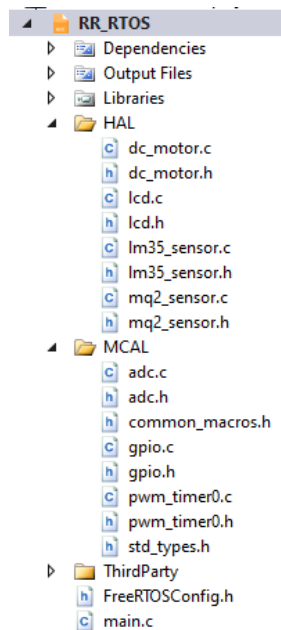


Figure 6-8: Layering of Software Implemented in the Application.

### 6.3. Implementing FreeRTOS on the Application

To Create an FreeRTOS application we have to follow some steps:

- First create a folder called Third party that will contain the FreeRTOS files
- Then inside the FreeRTOS files there is three main files which is the include folder that contains the header files of the freeRTOS files that should be included in the directory.
- Then a folder for the license file
- Then the last folder that depends on the Micro controller, the compiler, the memory management chosen file and some source files for freeRTOS implementation.

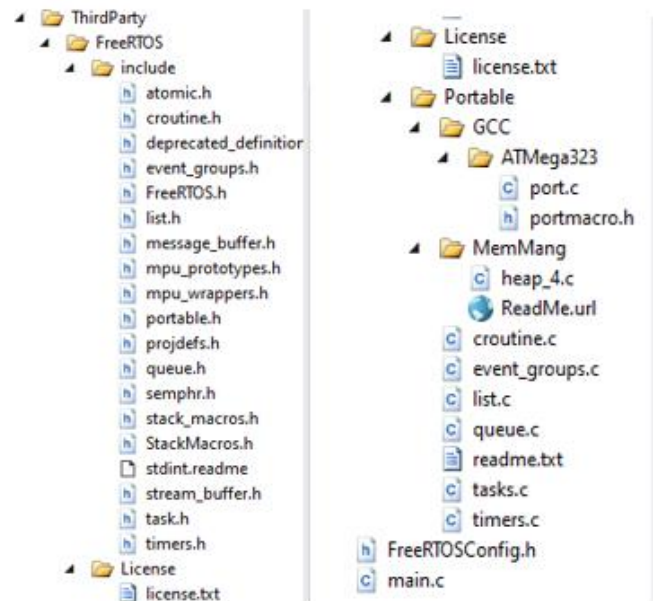


Figure 6-9: Chosen Files in Implementing the Code.

- The chosen compiler is GCC and the MC is ATmega323 which works fine with ATmega32 family too.
- In memory management folder the author choose heap 4 from 4 more different files each file have some advantage that makes this file better than the rest of files as heap 4 file has many advantages such as: combining adjacent free memory areas to avoid memory fragmentation. This file also provides the opportunity to store data at particular memory addresses [58].
- The final file that won't be in the FreeRTOS folder is "FreeRTOSConfig" as this file changes with each application.

### 6.4. Creating Task in FreeRTOS

FreeRTOS creates task by implanting this API "xTaskCreate()" that takes six inputs and returns "pdPASS" which means that this task is successfully implemented or "pdFAIL" which means this task could not be created due to lack of available stack memory [59].

The inputs given to the function is:

- The task function.
- The name wants to be known as in the stack or pc name.
- The stack size to be reserved this entered value in multiplied by word “ 4 bytes ”.
- pvParameters which is any input to be given or to be passed by reference to that function. In this application all the tasks are given “Null” value.
- Task priority.
- Handler is a label to be known to change anything of the task priority. Send their location.

This application consists of 4 main tasks which is: MQ2 sensor, LM35 sensor, implementing the appropriate reaction and final task displaying the fan state and the other sensors readings on the LCD. At the end of each task the author must put this API “ vTaskDelay(Ticks) ” to be transferred into the blocked state for a certain number of ticks this tick can be obtained by using this function “pdMS\_TO\_TICKS(time\_ms)” where the input is the time of the task to be blocked in seconds to transferred to ticks.

Table 6-3: Task Number and their Periodicity

Task number	Periodicity (ms)
Task 1 “Displaying data on LCD”	2000
Task 2 “Temperature Sensor”	1000
Task 3 “Motor”	250
Task 4 “Gas Sensor”	500

#### 6.4.1. Choosing Priority

Priority can be chosen by depending on the task’s importance or type. If the task is a sensor, the task must be at the highest priority so the application will not miss any readings. If the task is printing on LCD the priority can be least. If the task is to take appropriate reaction so their priority may be intermediate. And the priority differs from each scheduling algorithm and another.

#### 6.4.2. Putting the Task in the Block State

By using `vTaskDelay` and sending to this function a certain ticks the task will be locked in the blocked state and will not be able to be executed until the ticks is finished then the scheduler will check the priority to know which should be executed next.

#### 6.4.3. Putting the Task in the Suspended State

By using `vTaskSuspend` and send to this function the handler of the task wanted to be suspended or if the suspended task within the same function you could just send Null both methods will make the task in the suspended state and to make the task resume again, we can use `vTaskResume` and send to the function the task's handler. Where the handler is a pointer that points to the task's address "TCB" in the stack.

### 6.5. Implementing Different Scheduling Algorithms

In this application the author tried 2 different scheduling algorithms methods which are PB and RR. And in the results chapter will discuss more about which is better in performance and execution time.

#### 6.5.1. Priority Based

The higher priority is the first to be executed. The priority of the tasks of the application is as shown in table (6-4).

*Table 6-4: Tasks and their Priorities in PB Scheduling Algorithm*

Task number	Priority
Task 1 "Displaying data on LCD"	1
Task 2 "Temperature Sensor"	3
Task 3 "Motor"	2
Task 4 "Gas Sensor"	4

### 6.5.2. Round Robin

Round Robin have 2 possibilities either preemptive or non-preemptive. In this application the chosen scheduler is preemptive so the task can be interrupted if any of that task's priority increased. Also, the author make sure to that the interrupt and the time slicing is enabled. If the time between each tick interrupt is so small this will affect the application's performance and behavior due to the overhead and the context switching. If the time between each tick interrupt is so large it be the same as FCFS scheduling algorithm. The priority of the tasks of the application is as shown in table (6-5).

*Table 6-5: Tasks and their Priorities in RR Scheduling Algorithm.*

Task number	Priority
Task 1 "Displaying data on LCD"	2
Task 2 "Temperature Sensor"	2
Task 3 "Motor"	2
Task 4 "Gas Sensor"	2

### 6.5.3. Priority Based Round Robin

Priority based round robin scheduling algorithm is basically a combination between priority based and round robin scheduling algorithms. This algorithm takes advantage of RR and PB in a single scheduling algorithm. The priority of the tasks of the application is as shown in table (6-6).

*Table 6-6: Tasks and their Priorities in PBRR Scheduling Algorithm.*

Task number	Priority
Task 1 "Displaying data on LCD"	2
Task 2 "Temperature Sensor"	3
Task 3 "Motor"	2
Task 4 "Gas Sensor"	3



## 6.6. Hardware Implementation

Table (6-6) shows the components pins and where the hardware is connected to the MC.

Table 6-7: Components and the Connected Pins to the MC.

Component Pins		Pins in MC
LM35	Vin	VCC (5v)
	ground	GND
	Vout	A2
MQ2	Vin	VCC
	ground	GND
	Vanalog	A0
LCD	Rs	B7
	Rw	GND
	E	B6
	D0-D7	D0-D7
	VSS	GND
	VDD	VCC
	VEE	Variable rheostat
L298N	IN1	B0
	IN2	B1

## 6.7. Debugger

The author used Atmel ICE debugger to debug and test the codes and to get the results of the execution time of each task. The debugger was not available in Egypt so the author had to import it from USA. Also, the speed of the debugger should be less than or equal 0.25 of the system frequency which means 250 KHz as the operating frequency of this application is internal oscillator = 1 Mhz.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

The debugger connection is based on JTAG communication protocol which is the abbreviation for Joint Test Action Group. JTAG is a used to debug and program the MC. JTAG offers a standardized mechanism, enabling programmers to make use of the on-chip developing and debugging capabilities of MCs, triggering breakpoints, watching as code is executed, test and validate the functioning of their applications. JTAG can be connected to many devices. As shown in figure(6-11) but all the devices must share the same ground and Vcc [60]. The debugger will not be able to work with the microcontroller with the connection and the USB only the MC must be connected to an external power supply source. The author tried connecting a 9v battery series with diodes “IN5338B” to get 5v and tried to connect voltage regulator “7805” but both gives 5v volt measured by using Avometer but very low current that cannot let the debugger operates with the circuit. The results of the debugger will be shown in the next chapter.

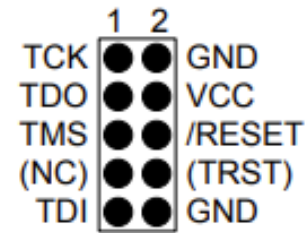


Figure 6-10: AVR JTAG pins.

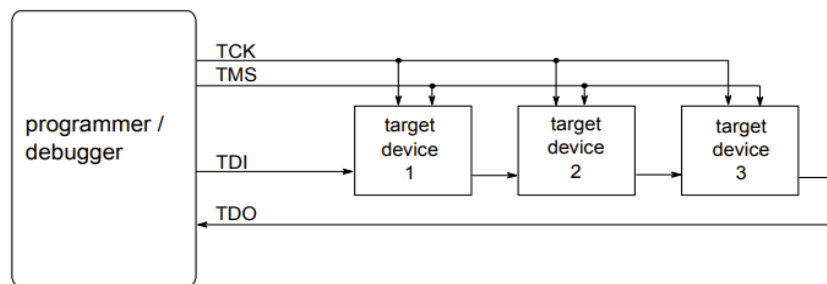


Figure 6-11: JTAG Connection with Multiple Devices [60].

Table (6-7), figure (6-10) shows the connection pins of JTAG with ATmega32A. where:

- TCK is test clock pin.
- TMS is test mode select
- TDI is test data input
- TDO is test data output

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 6-8: JTAG Connection with MC.

Component Pins		Pins in MC
Atmel ICE JTAG connection	TDI	C5
	TDO	C4
	TMS	C3
	TCK	C2
	Vcc	VCC
	GND	GND

## Chapter 7: Results

The results is checked by two methods software simulations which the author used proteus to ensure the functionality of the desired performance of the application and hardware implementation using Atmel ICE debugger.

### 7.1. Simulation on Proteus

The author have tried the simulation in proteus for each code first without RTOS then while using priority based then while using round robin with 3 different time slicing (0.1 sec, 10m sec, 100 $\mu$  sec) and by using priority based round robin.

Figure (7-1/ 7-2) shows the connections of the application parts with the MC and display the results on the LCD. Where the LEDs is just for illustration when the task begin to execute until the task finishes execution and then shows the on and off states and the execution on the oscilloscope.

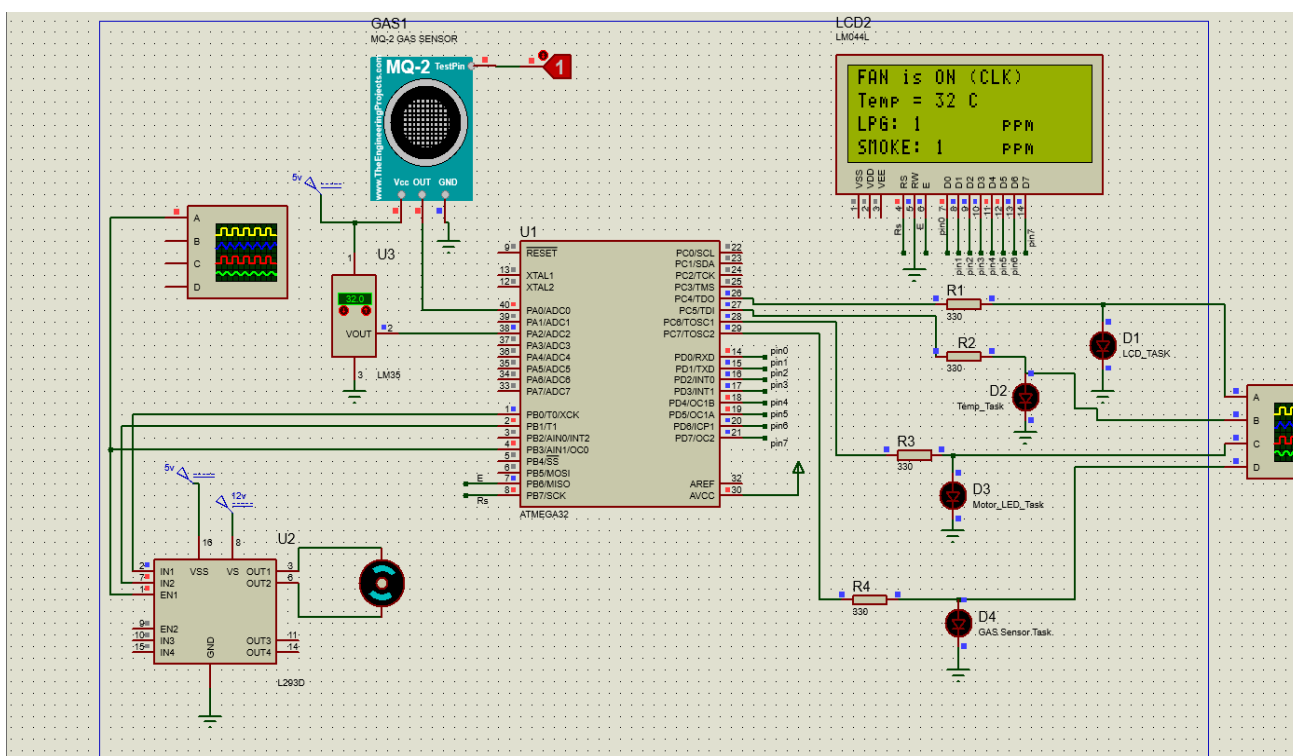


Figure 7-1: Proteus Simulation while the Temperature more than 30.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

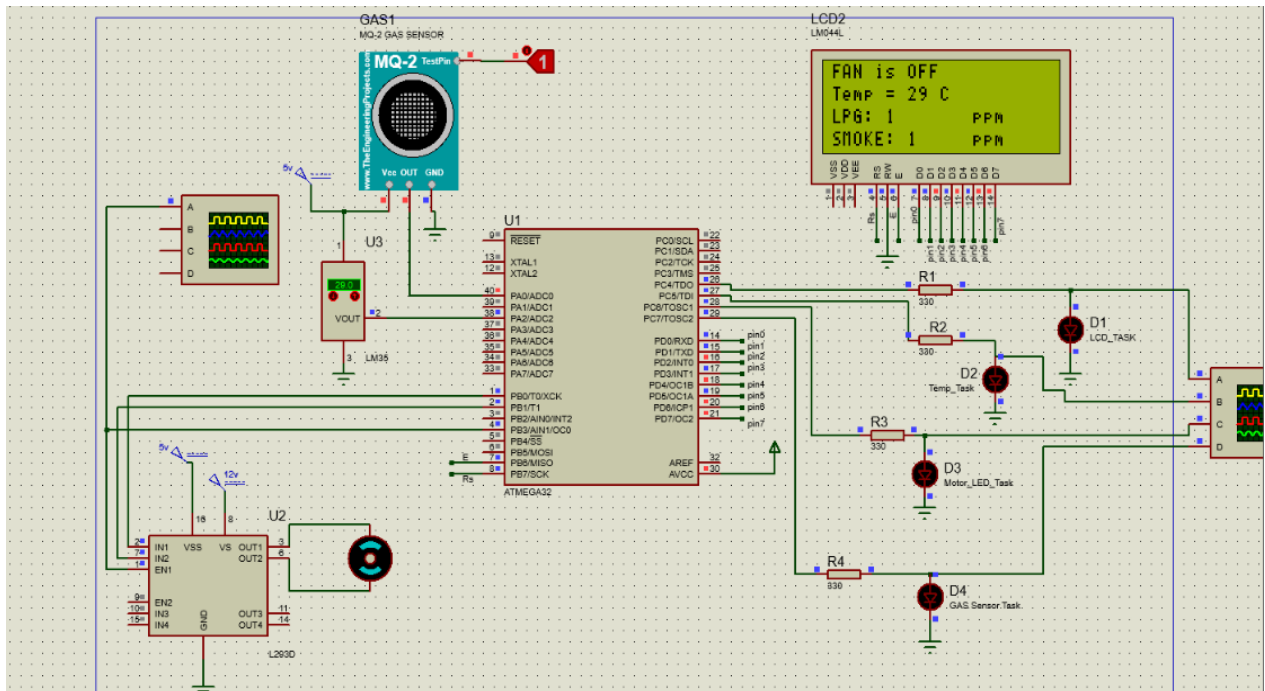


Figure 7-2: Proteus Simulation while the Temperature less than 30.

Figure (7-3) shows the PWM signal that enters the motor which begin in the simulation when the temperature is bigger than 30 °C.

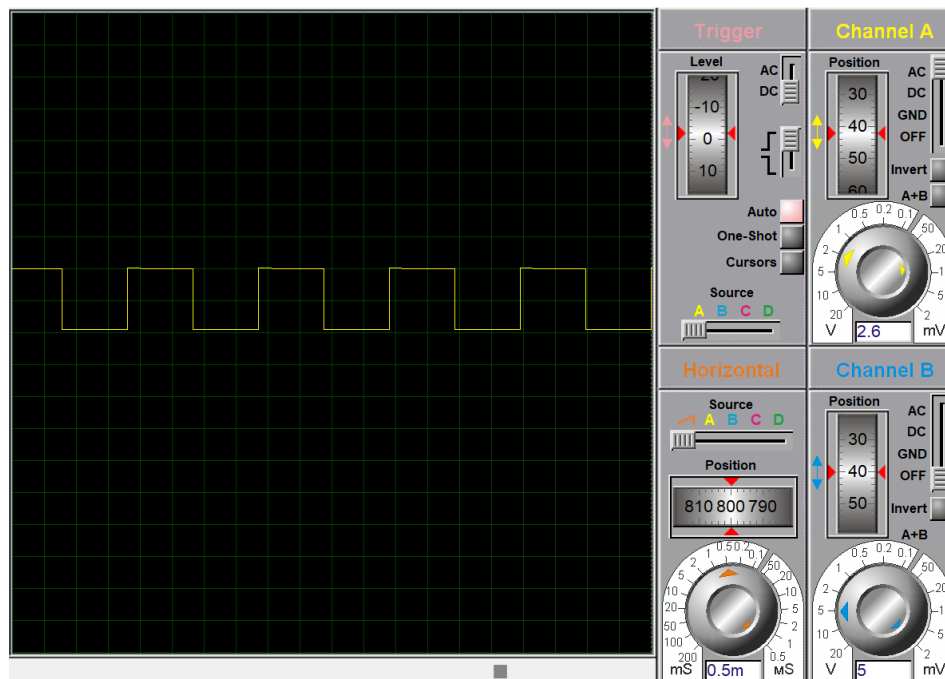
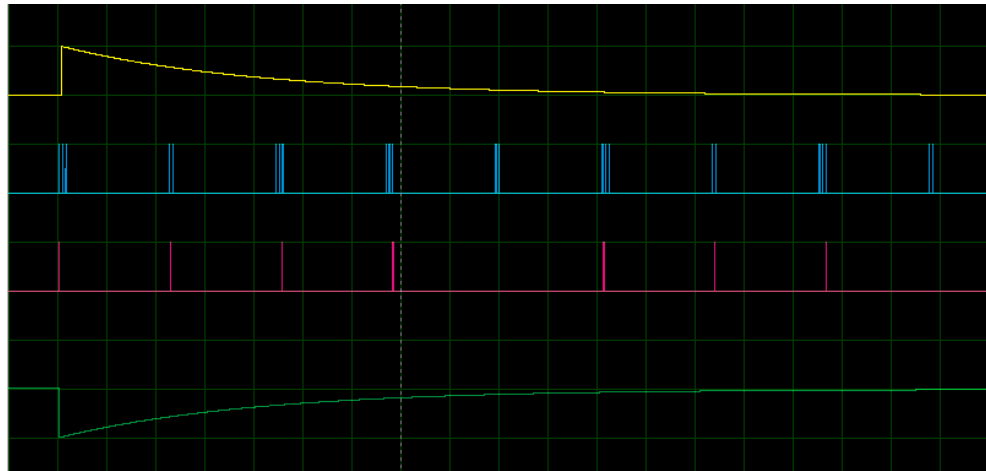


Figure 7-3: PWM Signal 50%.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Figure (7-4) shows the execution time of each task on Proetus's oscilloscope. Where "channel a" the yellow color indicates task one which is LCD printing, the blue line "channel b" indicates task two which is temperature sensor, the red line "channel c" indicates the motor task and the green line "channel d" shows the execution of gas sensor task. This was implemented on the priority-based scheduling algorithm.



*Figure 7-4: PB Task Execution on Proteus.*

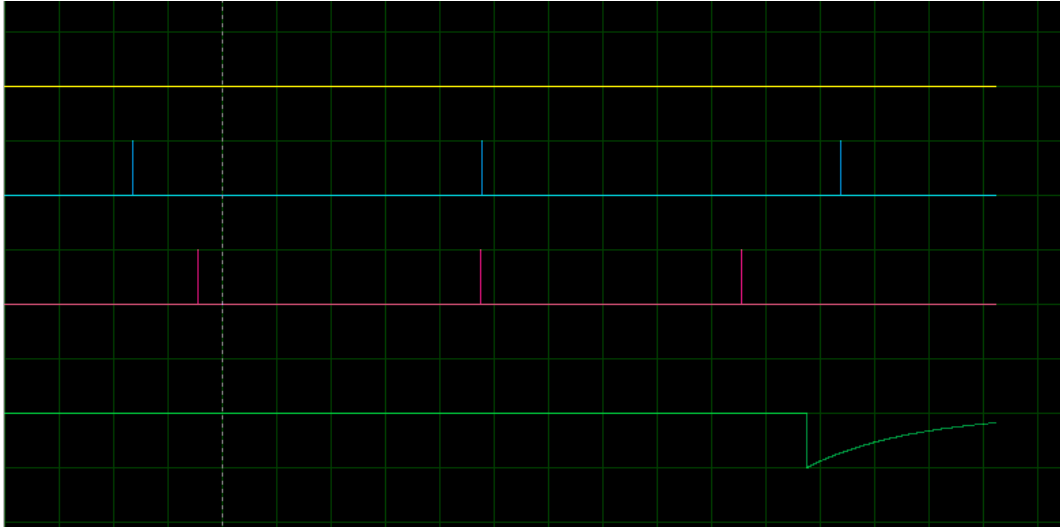
Figure (7-5) shows the execution time of each task on Proetus's oscilloscope. Where the same channel distribution as figure (7-4). The task execution shown in this figure was implemented based on round robin scheduling algorithm with time slicing (1/10).



*Figure 7-5: RR Task Execution with Time Slicing (1/10) on Proteus.*

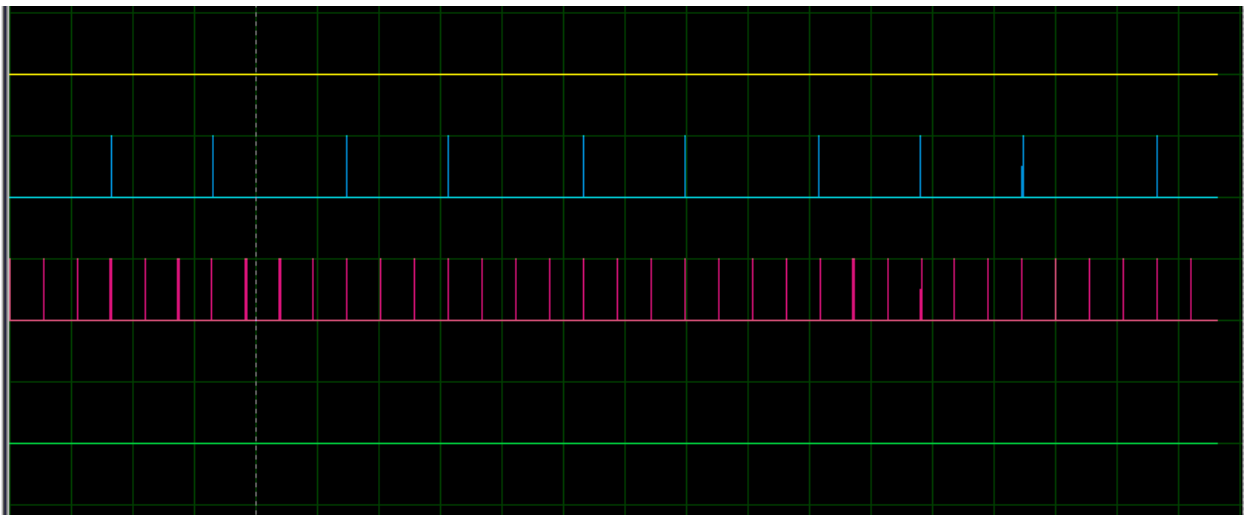
## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Figure (7-6) shows the execution time of each task on Proteus's oscilloscope. Where the same channel distribution as figure (7-4). The task execution shown in this figure was implemented based on round robin scheduling algorithm with time slicing (1/100).



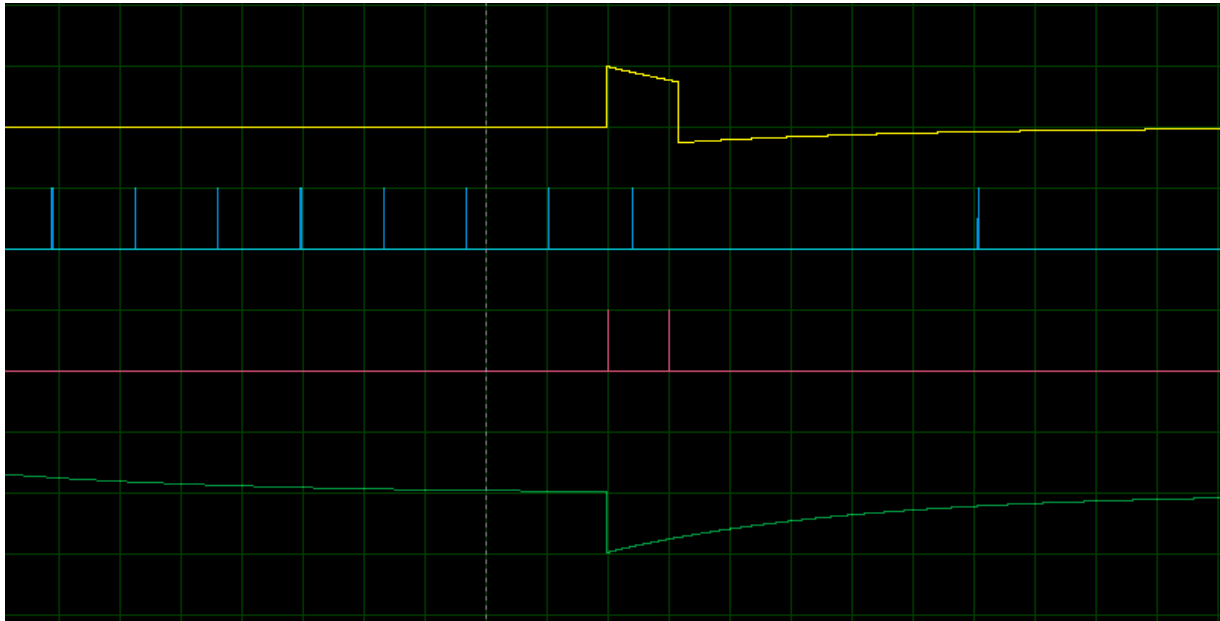
*Figure 7-6: RR Task Execution with Time Slicing (1/100) on Proteus.*

Figure (7-7) shows the execution time of each task on Proteus's oscilloscope. Where the same channel distribution as figure (7-4). The task execution shown in this figure was implemented based on round robin scheduling algorithm with time slicing (1/10000).



*Figure 7-7: RR Task Execution with Time Slicing (1/10000) on Proteus.*

Figure (7-8) shows the execution time of each task on Proetus's oscilloscope. Where the same channel distribution as figure (7-4). The task execution shown in this figure was implemented based on priority based round robin scheduling algorithm with time slicing (1/100).



*Figure 7-8: PBRR Scheduling Algorithm Execution Time.*

## 7.2. Debugging the Hardware

The author have tried the hardware by using Atmel ICE debugger for each code first without RTOS then while using priority based then while using round robin with 3 different time slicing (0.1 sec, 10m sec, 100 $\mu$  sec) and by using priority based round robin. The debugger provides the developer with so many powerful tools to see each step of the execution, even the assembly code as shown in figure (7-9).



## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Disassembly

Address: vGas\_Task

Viewing Options

- ☐ Show code bytes
- ☒ Show source code
- ☒ Show line numbers
- ☒ Show address
- ☒ Show symbol names

```

000003EA CALL 0x0000017D    Call subroutine
          96:    DcMotor_Init();                /* Initialize DC Motor */
000003EC CALL 0x00000107    Call subroutine
          97:    ADC_init(&ADC_Configurations);    /* Initialize ADC driver */
000003EE MOVW R24,R28    Copy register pair
000003EF ADIW R24,0x01    Add immediate to word
000003F0 CALL 0x0000045F    Call subroutine
          100:   SREG |= (1<<7);
000003F2 IN R24,0x3F    In from I/O location
000003F3 ORI R24,0x80    Logical OR with immediate
000003F4 OUT 0x3F,R24    Out to I/O location
          109:   status = xTaskCreate(vLcd_Task, "Lcd_Task", 100, NULL, LCD_Priority, &Lcd_Handler); //180
000003F5 MOV R0,R31    Copy register
000003F6 LDI R31,0x82    Load immediate
000003F7 MOV R14,R31    Copy register
000003F8 LDI R31,0x07    Load immediate
000003F9 MOV R15,R31    Copy register
000003FA MOV R31,R0    Copy register
000003FB LDI R16,0x01    Load immediate
000003FC LDI R18,0x00    Load immediate
000003FD LDI R19,0x00    Load immediate
000003FE LDI R20,0x64    Load immediate
000003FF LDI R21,0x00    Load immediate
00000400 LDI R22,0xD5    Load immediate
00000401 LDI R23,0x00    Load immediate
00000402 LDI R24,0x05    Load immediate
00000403 LDI R25,0x03    Load immediate
00000404 CALL 0x00000A7D    Call subroutine
  
```

Figure 7-9: Assembly Conversion Provided by the Debugger.

Figure (7-10) shows the number of tasks which is 5, the 4 that is implemented by the author and the fifth is the idle task the have priority of zero. Also shows the tick account from 0 to 0x FFFF which equals 65535. Also, shows the ready priority that should be executed next. And shows the top executed priority in the application if any tick is missed the counter will be increased. The number of overflows increases when the ticks reaches the maximum which is 65535.

System	
Name	Value
Number of Tasks:	5
Stack Growth Direction:	Downward
Tick Count:	0
Top Used Priority:	4
Top Ready Priority:	4
Scheduler Suspended:	No
Scheduler Running:	Yes
Missed Ticks:	NA
Number of Overflows:	0
Next Task Unblock Time:	There is no blocked task in the system
Number of Tasks To Be Terminated:	NA
Number of Priority Levels:	0

Figure 7-10: FreeRTOS Extensions on Atmel Studio.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Figure ( 7-11) shows the task handler which is the address of the TCB in the stack memory. The TCB contains the number of TCB which means the first task written is 1 and second is 2 etc, also TCB contains a pointer that points to the top of stack. Also shows the priority and the task name. The same analysis has been done to the RR with different time scheduling and the PBRR.

▲ (Lcd_Handler)	{struct tskTaskControlBlock{data}@0x01a6}	struct tskTaskControlBlock{data}@0x01a6
pxTopOfStack	0x017b	StackType_t*(data)@0x01a6
xStateListItem	{ListItem_t{data}@0x01a8}	ListItem_t{data}@0x01a8
xEventListItem	{ListItem_t{data}@0x01b2}	ListItem_t{data}@0x01b2
uxPriority	1	UBaseType_t{data}@0x01bc
pxStack	0x013e	StackType_t*(data)@0x01bd
pcTaskName	0x01bf	char[12]{data}@0x01bf
uxTCBNumber	1	UBaseType_t{data}@0x01cb
uxTaskNumber	0	UBaseType_t{data}@0x01cc
ulNotifiedValue	0x01cd	uint32_t[1]{data}@0x01cd
ucNotifyState	0x01d1	uint8_t[1]{data}@0x01d1
Temp_Sensor_Handler	0x023e	TaskHandle_t{data}@0x077c
▲ (Temp_Sensor_Handler)	{struct tskTaskControlBlock{data}@0x023e}	struct tskTaskControlBlock{data}@0x023e
pxTopOfStack	0x0213	StackType_t*(data)@0x023e
xStateListItem	{ListItem_t{data}@0x0240}	ListItem_t{data}@0x0240
xEventListItem	{ListItem_t{data}@0x024a}	ListItem_t{data}@0x024a
uxPriority	3	UBaseType_t{data}@0x0254
pxStack	0x01d6	StackType_t*(data)@0x0255
pcTaskName	0x0257	char[12]{data}@0x0257
uxTCBNumber	2	UBaseType_t{data}@0x0263
uxTaskNumber	0	UBaseType_t{data}@0x0264
ulNotifiedValue	0x0265	uint32_t[1]{data}@0x0265
ucNotifyState	0x0269	uint8_t[1]{data}@0x0269
Dc_motor_Handler	0x02d6	TaskHandle_t{data}@0x077e
▲ (Dc_motor_Handler)	{struct tskTaskControlBlock{data}@0x02d6}	struct tskTaskControlBlock{data}@0x02d6
pxTopOfStack	0x02ab	StackType_t*(data)@0x02d6
xStateListItem	{ListItem_t{data}@0x02d8}	ListItem_t{data}@0x02d8
xEventListItem	{ListItem_t{data}@0x02e2}	ListItem_t{data}@0x02e2
uxPriority	2	UBaseType_t{data}@0x02ec
pxStack	0x026e	StackType_t*(data)@0x02ed
pcTaskName	0x02ef	char[12]{data}@0x02ef
uxTCBNumber	3	UBaseType_t{data}@0x02fb
uxTaskNumber	0	UBaseType_t{data}@0x02fc
ulNotifiedValue	0x02fd	uint32_t[1]{data}@0x02fd
ucNotifyState	0x0301	uint8_t[1]{data}@0x0301
Gas_Sensor_Handler	0x036e	TaskHandle_t{data}@0x0780
▲ (Gas_Sensor_Handler)	{struct tskTaskControlBlock{data}@0x036e}	struct tskTaskControlBlock{data}@0x036e
pxTopOfStack	0x0343	StackType_t*(data)@0x036e
xStateListItem	{ListItem_t{data}@0x0370}	ListItem_t{data}@0x0370
xEventListItem	{ListItem_t{data}@0x037a}	ListItem_t{data}@0x037a
uxPriority	4	UBaseType_t{data}@0x0384
pxStack	0x0306	StackType_t*(data)@0x0385
pcTaskName	0x0387	char[12]{data}@0x0387
uxTCBNumber	4	UBaseType_t{data}@0x0393
uxTaskNumber	0	UBaseType_t{data}@0x0394
ulNotifiedValue	0x0395	uint32_t[1]{data}@0x0395
ucNotifyState	0x0399	uint8_t[1]{data}@0x0399

Figure 7-11: Task Handlers and TCB of Each Task.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Figure ( 7-12 ) shows the registers found in the ATmega32A and their bits configuration and the value of the flags or the registers all these values and bits could be changed easily through this interface also the author could keep an eye on the timer counter values.

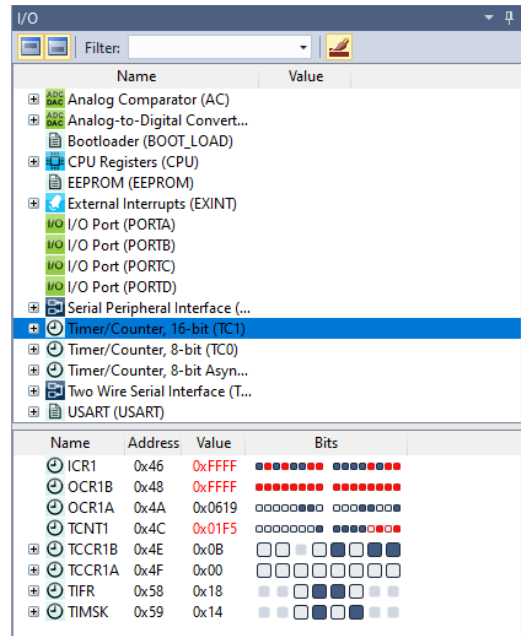


Figure 7-12: Registers and their Bits.

Table (7-1) shows the execution time of every task and every scheduling algorithm by using the xTaskGetTickCount the gives the ticks of made till now in each task and from the extension in Atmel studio the author could get the total number of ticks.

First, the application without FreeRTOS could not get the ticks number as the application is not implemented as tasks and not by RTOS and xTaskGetTickCount is a RTOS API.

By using this equation, we can get the value of the time:

$$\text{time of task} = \text{time of one tick} * \text{no. of ticks to implement this task}$$

For the PB values was:

- 1 Tick = 1/1000
- T1 = 9064 Ticks, so T1 = (1/1000) \* 9064 = 9.064 sec
- T2 = 1 Ticks, so T1 = (1/1000) \* 1 = 0.001 sec
- T3 = 2 Ticks, so T1 = (1/1000) \* 2 = 0.002 sec
- T4 = 4972 Ticks, so T1 = (1/1000) \* 4972 = 4.972 sec
- Total = 14069 Ticks, so total = (1/1000) \* 14069 = 14.069 sec

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

For the RR values was:

- 1 Tick =  $1/10$
- T1 = 10 Ticks, so  $T1 = (1/10) * 10 = 1 \text{ sec}$
- T2 = 4 Ticks, so  $T1 = (1/10) * 4 = 0.4 \text{ sec}$
- T3 = 8 Ticks, so  $T1 = (1/10) * 8 = 0.8 \text{ sec}$
- T4 = 146 Ticks, so  $T1 = (1/10) * 146 = 14.6 \text{ sec}$
- Total = 148 Ticks, so total =  $(1/10) * 148 = 14.8 \text{ sec}$

For the RR values was:

- 1 Tick =  $1/100$
- T1 = 69 Ticks, so  $T1 = (1/100) * 69 = 0.69 \text{ sec}$
- T2 = 4 Ticks, so  $T1 = (1/100) * 4 = 0.04 \text{ sec}$
- T3 = 8 Ticks, so  $T1 = (1/100) * 8 = 0.08 \text{ sec}$
- T4 = 738 Ticks, so  $T1 = (1/100) * 738 = 7.38 \text{ sec}$
- Total = 745 Ticks, so total =  $(1/100) * 745 = 7.45 \text{ sec}$

For the RR values was:

- 1 Tick =  $1/10000$
- T1 = 69 Ticks, so  $T1 = (1/10000) * 69 = 69.7 \text{ sec}$
- T2 = 4 Ticks, so  $T1 = (1/10000) * 4 = 0.0688 \text{ sec}$
- T3 = 8 Ticks, so  $T1 = (1/10000) * 8 = 0.1127 \text{ sec}$
- T4 = 146 Ticks, so  $T1 = (1/10000) * 146 = 131.4224 \text{ sec}$
- Total = 148 Ticks, so total =  $(1/10000) * 148 = 131.44 \text{ sec}$

For the PBRR values was:

- 1 Tick =  $1/100$
- T1 = 34 Ticks, so  $T1 = (1/100) * 34 = 0.34 \text{ sec}$
- T2 = 1 Ticks, so  $T1 = (1/100) * 1 = 0.01 \text{ sec}$
- T3 = 2 Ticks, so  $T1 = (1/100) * 2 = 0.02 \text{ sec}$
- T4 = 366 Ticks, so  $T1 = (1/100) * 366 = 3.66 \text{ sec}$
- Total = 405 Ticks, so total =  $(1/100) * 405 = 4.05 \text{ sec}$

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 7-1: Execution Time for Tasks with Different Scheduling Algorithms

Task number	Without RTOS	Priority Based (sec)	Round Robin			PBRR (sec)
			Time Slice = 1/10 (sec)	Time Slice = 1/100 (sec)	Time Slice = 1/10000 (sec)	
Task 1	--	9.064	1	0.69	69.7	0.34
Task 2	--	0.001	0.4	0.04	0.0688	0.01
Task 3	--	0.002	0.8	0.08	0.1127	0.02
Task 4	--	4.972	14.6	7.38	131.4224	3.66
Total execution time	--	14.069	14.8	7.45	131.44	4.05

As Table (7-1) shows the best scheduling algorithm for this application is priority based round robin that shows perfect execution time for the total application and each task individually. After this scheduling algorithm comes the normal RR after if comes the large time slice which works as FCFS after this scheduling algorithm the normal priority comes and finally the worst execution was the RR with very small time slice because of the overhead and the context switching.

Table (7-2) shows the total execution time of the same application but using different scheduling algorithms by using a timer 1.

The execution time of each task was a bit difficult as all task enters to each other at the RR scheduling algorithm

By using equation 3 that is explained in chapter 5 we can get the total executed time for the whole application.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

For the Without RTOS the values was:

- Global counter = 74, TCNT1 = 0x0470 = 1136.

By using equation (3)

$$time\ executed = 74 * 65.536 * 10^{-3} + 1136 * 10^{-6} = 4.8508\ sec$$

For the PB scheduling algorithm the values was:

- Global counter = 3, TCNT1 = 0x000C = 12.

By using equation (3)

$$time\ executed = 3 * 65.536 * 10^{-3} + 12 * 10^{-6} = 0.19662\ sec$$

For the RR scheduling algorithm time slicing = 1/10, The values was:

- Global counter = 1 , TCNT1 = 0x0593 = 1427.

By using equation (3)

$$time\ executed = 1 * 65.536 * 10^{-3} + 1427 * 10^{-6} = 0.066963\ sec$$

For the RR scheduling algorithm time slicing = 1/100, the values was:

- Global counter = 1 , TCNT1 = 0x0083 = 131.

By using equation (3)

$$time\ executed = 1 * 65.536 * 10^{-3} + 131 * 10^{-6} = 0.065667\ sec$$

For the RR scheduling algorithm time slicing = 1/10000, the values was:

- Global counter = 20 , TCNT1 = 0x9B92 = 39826 .

By using equation (3)

$$time\ executed = 20 * 65.536 * 10^{-3} + 39826 * 10^{-6} = 1.350546\ sec$$

For the PBRR scheduling algorithm the values was:

- Global counter = 1, TCNT1 = 0x0081 = 129 .

$$time\ executed = 1 * 65.536 * 10^{-3} + 129 * 10^{-6} = 0.065665\ sec$$

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

Table 7-2: Execution Time of Different Scheduling Algorithms by using Timer.

Task number	Without RTOS (sec)	Priority Based (sec)	Round Robin			PBRR (sec)
			Time Slice = 1/10 (sec)	Time Slice = 1/100 (sec)	Time Slice = 1/10000 (sec)	
<b>Total execution time</b>	4.5808	0.19662	0.066963	0.065667	1.350546	0.065665

Table (7-2) shows that the best scheduling algorithm is PBRR then RR with normal time slicing then the RR with large time slicing acting like FCFS then the normal priority based. And of course, we can see the difference in execution time between with or without using RTOS

Table 7-3: Memory Consumption With or Without RTOS and with Different Scheduling Algorithms.

Task number		Without RTOS (bytes ,% Full)	Priority Based (bytes ,% Full)	Round Robin (bytes ,% Full)	PBRR (bytes ,% Full)
<b>Memory Consumption</b>	<b>ROM</b>	6112, 18.7%	10244, 31.3%	10334, 31.5%	10316, 31.5%
	<b>RAM</b>	163, 8.0%	1852, 90.4%	1842, 89.9%	1840, 89.9%

Table (7-3) shows the memory consumption of RAM “Data Memory” and ROM “Program Memory” with and without RTOS and with different types of RTOS.

### 7.3. Simulating by using Simso

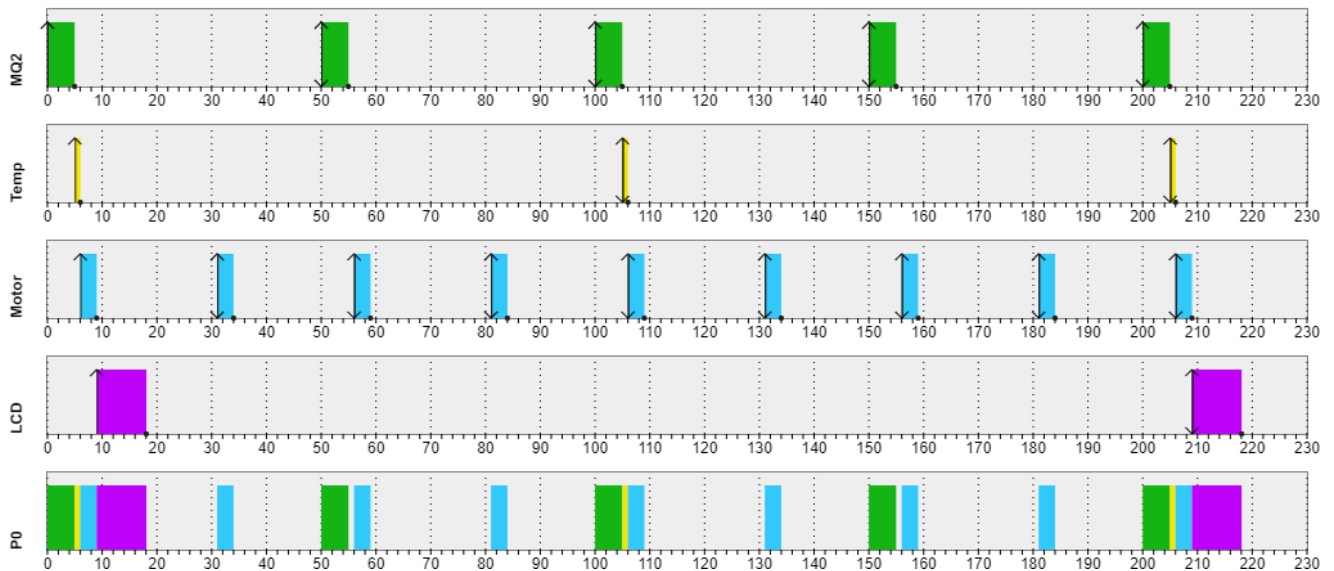


Figure 7-13: Simulation Shows the Task Execution Based on PB Scheduling Algorithm.

Figure (7-13) shows the execution of the 4 tasks based on PB scheduling algorithm on Simso simulator after knowing the execution time of each task from the debugger, entering the priority of each task and entering the periodicity of the task normalized by 10. Also, Simso provides the CPU usage value which equals to 31.734 %.

### 7.4. Segger SystemView

The author tried to implement the application with Segger SystemView to check the performance but could not find the configuration files for the AVR MC and found only the files for ARM MC. So, the author could not proceed with checking the performance with this method.

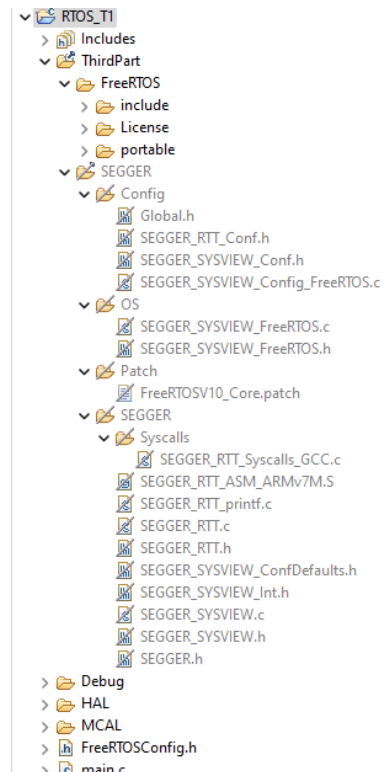


Figure 7-14: Segger SystemView Implementation with the Application.



## Chapter 8: Conclusion

In conclusion, This research begins at first with explaining the basics differences between RTOS and GPOS and their applications, latency and preemptive. Then explains the RTOS architecture, software types, RT types such as: hard, firm and soft and their differences, examples and the effect of missing a deadline then the advantages and disadvantages of using the RTOS. Then the author talks about the RTOS operation, the task states (ready / running / blocked ), the task control blocks that contains various data on the created task such as: task id, priority the stack size, status and pointer to the stack, then the author dig deeply into the various scheduling algorithms and their types either preemptive or non-preemptive and the static or the dynamic scheduling algorithms by providing example for each and explaining their advantages, disadvantages and their definitions. Examples of the explained scheduling algorithms: Round Robin, Priority Based, Least slack First, Earliest Deadline First, First Come First Serve, Shortest Job First, Priority Based Round Robin and non-preemptive Deadline First and then the author talk about technical keywords related to RTOS and their meaning such as: Deadlocks, Mutex and Binary or counter semaphore .

Then the author shows the software's that can be used to measure the performance, which is Segger SystemView “ single shot and continuous shot and differences between them”, Percepio Tracealyzer, Simso Simulator and Keil MDK that is unfortunately works with ARM MC only and explained the working principle of each one of them in more details. Also, the criteria of the performance that may be Evaluated: task execution time, total time execution, context switching, interrupt latency, stack usage and heap memory management, total memory consumption, power consumption, WCET and CPU usage.

The author provided analytical analysis of the performance through the Atmel ICE debugger by using FreeRTOS APIs and another method by using timer. The Problem that this research would like to shed light on is the increase of death and incidents of LPG burns and the patients within the ages of 21 and 60 experienced LPG burns the most frequently (73.85 %) and almost 81.03 % of this percentage was related to gas leaks and neglecting the kitchen (2 %). So, the author made a solution which is make an application by using FreeRTOS that is implemented on ATmega32A which contain a couple of sensors (LM35 and MQ2) with a lcd to display the

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

sensors readings and take the appropriate response by using dc motor works as ventilating fan. The author shows the PPM values of CO and the time before critical symptoms.

Moreover, the author tried 3 different scheduling algorithms which was PB, RR with different time slicing ( 0.1 sec, 0.01 sec,  $100 * 10^{-6}$  sec ) and PBRR (0.01 sec). The results shows that the best scheduling algorithm in time execution was PBRR with time = 0.0656 sec while the rest less than that and without RTOS takes about 4.581 sec however, the memory consumption of the application without RTOS was: RAM = 163 Bytes which means 8 % full, ROM = 6112 Bytes which means 18.7 % while the PBRR memory consumption was RAM = 1840 Bytes which means 89.9 % full, ROM = 10316 Bytes which means 31.5 % also there is a simulation that shows the execution of the tasks made by using Simso for PB only. In the future the author will try to implement these scheduling algorithms with different RTOS and on a multicore MC.

## References

- [1] S. Canbaz and G. Erdemir, "Performance analysis of real-time and general-purpose operating systems for path planning of the multi-robot systems," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 1, pp. 285-292, 2022.
- [2] P. Hambarde, R. Varma and S. Shivani, "THE SURVEY of REAL TIME OPERATING SYSTEM: RTOS," in *International Conference on Electronic Systems, Signal Processing and Computing Technologies*, 2014.
- [3] G. A. Ismael, A. A. Salih, A. AL-Zebari, N. Omar, K. J. Merceedi, A. J. Ahmed, N. O. M. Salim, S. S. Hasan, S. F. Kak, . I. . M. Ibrahim and H. M. Yasin, "Scheduling Algorithms Implementation for Real Scheduling Algorithms Implementation for Real," *Asian Journal of Research in Computer Science*, pp. 35-51, 2021.
- [4] R. Luna and S. A. Islam, "Security and Reliability of Safety-Critical RTOS," A *SPRINGER NATURE journal*, 2021.
- [5] P. Hambarde, R. Varma and S. Jha, "THE SURVEY of REAL TIME OPERATING SYSTEM: RTOS," in *International Conference on Electronic Systems, Signal Processing and Computing Technologies*, 2014.
- [6] T. Nguyen, B. Anh and S.-L. Tan, "REAL-TIME OPERATING SYSTEMS FOR SMALL MICROCONTROLLERS," 2009.
- [7] P. Prasad, K. Bobade, K. Raikar and M. Pande, "An Approach for Real-Time Nested Lock," in *Proceedings of the International Conference on Communication and Electronics Systems (ICCES 2018)*, 2018.
- [8] Y. Xing, J. Zhu, Y. Fu, Y. Zhang and Q. Qiao, "Embedded Toxic Gas Monitor Based on MCOS-II," in *International Conference on Mechatronics and Automation*, 2019.
- [9] G. A. Chelaru and F. C. Braescu , "FreeRTOS Based Aquarium Monitoring and Maintenance Embedded System," in *25th International Conference on System Theory, Control and Computing (ICSTCC)*, 2021.
- [10] J. J. Palatty, S. P. Chandra and S. P, "Performance Analysis of FreeRTOS Based Video Capture System," in *Third International Conference on Electronics Communication and Aerospace Technology [ICECA]*, 2019.
- [11] J. Arm, O. Baštán, O. Mihálik and Z. Bradáč, "Measuring the Performance of FreeRTOS on ESP32 Multi-Core," in *IFAC PapersOnLine*, 2022.
- [12] T. S. Gunawan, M. F. Sabar, H. Nasir, M. Kartiwi and S. M. Motakabber, "Development of Smart Chicken Poultry Farm using RTOS on Arduino," in *6th International*

*Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, Kuala Lumpur, 2019.

- [13] B. Septian, M. Misbahuddin and F. Arkan, "FREERTOS BASED AIR QUALITY MONITORING SYSTEM USING SECURE INTERNET OF THINGS," *Jurnal Teknik Informatika (JUTIF)*, vol. III, no. 1, pp. 147-153, 2022.
- [14] C. A. SARMA, M. R. M. REDDY and G. S. REDDY, "REAL TIME OPERATING SYSTEM (RTOS) FOR EMBEDDED SYSTEM APPLICATION DESIGN," *IAETSD JOURNAL FOR ADVANCED RESEARCH IN APPLIED SCIENCES*, vol. 5, no. 5, pp. 181-184, 2018.
- [15] I. Ungurean and N. C. GAITAN, "Performance Analysis of Tasks Synchronization for Real Time Operating Systems," in *International Conference on DEVELOPMENT AND APPLICATION SYSTEMS*, Suceava, 2018.
- [16] T. D. Juhász , S. Pletl and L. Molnar , "A Method for Designing and Implementing a Real Time Operating System for Industrial Devices," in *International Symposium on Applied Computational Intelligence and Informatics*, Timișoara, 2019.
- [17] S. Kar and S. K. Singh, "Open Source Real Time Internet of Things and Free RTOS Based Temperature Monitoring System," *International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)*, vol. 11, no. 4, pp. 3927-3932, 2022.
- [18] M. Khomenko and O. Velihorskyi, "The Use of Percepio Tracealyzer for the Development of FreeRTOS-based Applications," in *International Scientific and Practical Conference "Theoretical and Applied Aspects of Device Development on Microcontrollers and FPGAs" (MC&FPGA-2020)*, 2020.
- [19] A. Akhoury, A. Ravi, K. Birla, S. Kalsi, R. Sarkar and S. Ghorai, "Design and Analysis of RTOS and Interrupt Based Data Handling System for Nanosatellites," 2019.
- [20] R. Edmaier, N. Ueter and J. J. Chen, "Segment-Level FP-Scheduling in FreeRTOS," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022.
- [21] G. Akgun and D. Gohringer, "Power-Aware Real-Time Operating Systems on Reconfigurable Architectures," in *31st International Conference on Field-Programmable Logic and Applications (FPL)*, Dresden, 2021.
- [22] J. Teraiya and A. Shah, "Analysis of Dynamic and Static Scheduling Algorithms in Soft Real-Time System with Its Implementation," 2020.

- [23] F. M. S. Nascimento and G. Lima, "Effectively Scheduling Hard and Soft Real-Time Tasks on Multiprocessors," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.
- [24] T. D. Putra, "Analysis of Priority Preemptive Scheduling Algorithm: Case Study," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 11, no. 1, 2022.
- [25] P. Sinha, B. Prabadevi, S. Dutta, D. N and N. Kumari , "Efficient Process Scheduling Algorithm using RR and SRTF," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Vellore, India, 2020.
- [26] H. M. Abu-Dalbouh, "A New Combination Approach to CPU Scheduling based on Priority and Round-Robin Algorithms for Assigning a Priority to a Process and Eliminating Starvation," (*IJACSA*) *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, pp. 541-546, 2022.
- [27] S. Zouaoui, L. Boussaid and A. Mtibaa, "Priority based round robin (PBRR) CPU scheduling algorithm," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 190-202, 2019.
- [28] M. G. Unguritu and T. C. Nichiștea, "Adaptive Real-Time Operating System in Automotive Multicore Embedded Systems," in *International Conference on System Theory, Control and Computing (ICSTCC)*, Iasi, 2021.
- [29] L. R. Lencioni, D. S. Loubach and O. Saotome, "A Methodology for Customization of a Real-Time Operating System for Embedded Systems," in *International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 2021.
- [30] M. Begum, O. Faruque, M. W. R. Miah and B. C. Das, "An Improved Safety Detection Algorithm Towards Deadlock Avoidance," in *IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Malaysia, 2020.
- [31] R. XU, L. Zhang and N. Ge, "Modeling and Timing Analysis for Microkernel-Based Real-Time Embedded System," vol. 7, pp. 39547-39563, 2019.
- [32] P. Raffek, P. Ulbrich and W. S. Preikschat, "Work-In-Progress: Migration Hints in Real-Time Operating Systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2019.
- [33] D. Mamone, A. Bosio, A. Savino, S. Hamdioui and M. Rebaudengo, "On the Analysis of Real-time Operating System Reliability in Embedded Systems," 2020.
- [34] A. Bosio, M. Rebaudengo and A. Savino, "Reliability assessment of FreeRTOS in Embedded Systems," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2022.

- [35] A. Serino and L. Cheng, "Real-Time Operating Systems for Cyber-Physical Systems: Current Status and Future Research," in *International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2020.
- [36] A. W. Alagalla and U. U. Rajapaksha, "Techniques of Enhancing Synchronization Efficiency of Distributed Real Time Operating Systems," in *2nd International Conference on Advanced Research in Computing (ICARC)*, 2022.
- [37] W. Nakano, Y. Shinohara and N. Ishiura, "Full Hardware Implementation of FreeRTOS-Based Real-Time Systems," in *TENCON 2021-2021 IEEE Region 10 Conference (TENCON)*, 2021.
- [38] J. Alhiyafi, "PERFORMANCE EVALUATION OF NON-PREEMPTIVE EARLIEST DEADLINE FIRST SCHEDULING TECHNIQUES," *ICIC EXPRESS LETTERS*, vol. 10, pp. 17-24, 2019.
- [39] T. D. Putra, "Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 9, no. 4, 2020.
- [40] J. Teraiya and A. Shah , "Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Bangalore, India, 2018.
- [41] I. Muller, F. Vargas, R. Severo, C. M. da Costa, A. Parraga and D. Motta , "Design and Test of the RT-NKE Task Scheduling Algorithm for Multicore Architectures," 2018.
- [42] B. S, C. MP and D. SN, "COMPREHENSIVE ANALYSIS OF CPU SCHEDULING ALGORITHMS," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 04, no. 09, pp. 180-185, 2022.
- [43] M. Riasetiawan, A. Ashari and Hayatunnufus, "Performance Analysis of FIFO and Round Robin Scheduling Process Algorithm in IoT Operating System for Collecting Landslide Data," in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 2020.
- [44] V. Prajapati, A. Shah and P. Balani, "Design of new scheduling algorithm LLF\_DM and its comparison with Existing EDF, LLF, and DM algorithms for periodic tasks," in *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*, Gujarat, India, 2013.
- [45] K. Abe, M. Yoo and . T. Yokoyama, "Aspect-Oriented Customization of the Scheduling Algorithm and the Resource Access Protocol of a Real-Time Operating System," in

*IEEE 16th International Conference on Computational Science and Engineering*, Tokyo, Japan, 2013.

- [46] "SEGGER System View," SEGGER, [Online]. Available: <https://www.segger.com/products/development-tools/systemview/technology/what-is-systemview/>.
- [47] "Tracealyzer Visual Runtime Insights," Percepio Sensing software, [Online]. Available: <https://percepio.com/tracealyzer/>.
- [48] "SimSo documentation," SimSo, [Online]. Available: <https://projects.laas.fr/simso/doc/introduction.html#what-is-simso>.
- [49] "Keil MDK," armDeveloper, [Online]. Available: <https://developer.arm.com/Tools%20and%20Software/Keil%20MDK>.
- [50] S. Zouaoui, L. Boussaid and A. Mtibaa, "Priority based round robin (PBRR) CPU scheduling algorithm," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 190-202, 2019.
- [51] R. Jin, P. Wu, J. K. Ho, X. Wang and C. Han, "Five-year epidemiology of liquefied petroleum gas Related burns," *BURNS*, vol. 44, pp. 210-217, 2018.
- [52] "How MQ2 Gas/Smoke Sensor Works? & Interface it with Arduino," Last Minute ENGINEERS, [Online]. Available: <https://lastminuteengineers.com/mq2-gas-senser-arduino-tutorial/>.
- [53] "Hazardous Substance Fact Sheet (Liquefied Petroleum Gas)," NJ Health, New Jersey, 2010.
- [54] "Hazardous Substance Fact Sheet (Carbon Monoxide)," NJ Health, New Jersey, 2016.
- [55] M. Goldstein, "Carbon Monoxide Poisoning," *Journal of EMERGENCY NURSING*, vol. 34, no. 6, pp. 538-542, 2008.
- [56] T. Struttman, A. Scheerer, T. S. Prince and L. A. Goldstein, "Unintentional Carbon Monoxide Poisoning From an Unlikely Source," *JABFP*, vol. 11, no. 6, pp. 481-484, 1998.
- [57] "Interface L298N DC Motor Driver Module with Arduino," Last Minute ENGINEERS, [Online]. Available: <https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/>.
- [58] "Memory Mangemnet," FreeRTOS, [Online]. Available: <https://www.freertos.org/a00111.html>.

- [59] R. Barry, "Mastering the FreeRTOS™ Real Time Kernel A Hands-On Tutorial Guide," 2016.
- [60] "Atmel-ICE," MICROCHIP, 2016. [Online]. Available: [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-ICE\\_UserGuide.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-ICE_UserGuide.pdf).
- [61] H. Wang, J. Wang, Z. Wang, W. Hu, L. Zeng, X. Xi and X. Wang, "Research on Real-time Analysis Method for Real-time Operating System," in *IEEE 5th International Conference on Electronic Information and Communication Technology (ICEICT)*, 2022.



## Appendix A

# Executive Summary: Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

## Abstract

This research paper discuss the Real Time Operating System and their architecture, types, software types, benefits, and drawbacks, as well as how they work, including task states and control blocks. This research goes into great detail to explain many scheduling algorithms, including Round Robin, Priority Based, Earliest Deadline First, First Come First Serve, and non-preemptive Deadline First. It also covers the performance monitoring tools and what should be monitored while measuring the performance. The problem that the authors shed light on is the increment of death and incidents of Liquified Petroleum Gas (LPG) burns and the patients within 21 to 60 experienced LPG burns the most frequently 73.85% and almost 81.03% of this percentage was related to gas leak and neglecting the kitchen 2%. So, the author made an application by using FreeRTOS with different scheduling algorithms PB, RR “with 3 different time slicing 1/10, 1/100, 1/10000” and PBRR that is implemented on ATmega32A which contain a couple of sensors with a lcd to display the sensors readings and take the appropriate response by using dc motor works as ventilating fan. The author shows the PPM values of CO and the time before critical symptoms begins. The results shows that the best scheduling algorithm in time execution was PBRR with time = 0.0656 sec while the rest take more time than that and without RTOS takes 4.581 sec however, The memory consumption of the application without RTOS was RAM of 8% full, and ROM of 18.7% full while the PBRR memory consumption was RAM of 89.9% full, of 31.5% full also there is a simulation that shows the execution of the tasks made by using Simso for PB scheduling algorithm only.

## Problem Statement

When designing a real time application meeting a time requirement “deadline” is essential to be taken into consideration as missing the deadline will affect the application’s

performance or even lead to a failure in the application's performance and may lead to a lot of problems or even a disaster that may affect a lot of people's life and puts their life on a stake. This is why the developer must design and implement a well-designed application that ensure not to miss any deadlines. Also, choosing the best scheduling algorithm that is suitable for this specific application is a bit of a challenging task regarding the execution time, Which task should be executed first and based on what as an example: EDF, FP, RR or other scheduling algorithms.

The LPG leakage is a serious problem as this gas is odorless and colorless and may lead to series injuries or death depends on the time and its intensity of the gas that the patient had exposed to. Also, the annual incidence of LPG-related burns was 10% of all burning injuries for the first four years (2011-2014), but it increased to 26.94% in just the fifth year. LPG-related burns were primarily caused by gas leaks (81.03%), followed by improper operation (7.69%) and carelessness in the kitchen (2.05%) [1, 2].

## Objective

The aim of this research is to design and implement a RT embedded system's application by using three different scheduling algorithms PB, RR with different time slicing (0.1,  $10 \times 10^{-3}$ ,  $100 \times 10^{-6}$ ) and PBRR and compare which is better in performance to detect any dangerous leakage gases such as LPG and  $CO_2$  or any abnormal in temperature in the kitchen then show it on screen and take an appropriate reaction which is by opening the ventilating fan in a certain direction or stop the ventilating fan.

## Brief Background

RTOS is considered a software layer found between Application layer and low-level drivers that is not related to the hardware which is: LCD, Sensors, Motors and keypad. The Hardware layer is related with the MC interfaces such as: GPIO, timer and ADC [1].

RTOS have 3 main types hard, firm and soft. Hard RT the execution of the application must meet a specific time not before or after the deadline any delay means a failure in execution. Soft RT if any delay happens it is acceptable but the execution will be affected. Firm RT may miss few deadlines can be acceptable more than that will lead to system failure [3].

There is many advantages for using RTOS such as: synchronization and safety are improved by using a global variable or a semaphore which have 2 types: binary and counting

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

semaphore [4]. By the same concept semaphores can be used to detect bugs and RTOS offers layering architecture that makes the programmer organize and produce cleaner code [4].

However, there is many disadvantages of using RTOS [4]. such as Difficulty and complexity in designing, Consumes more memory and the programmer must be aware of the execution time to take it into his consideration while designing the task flow and the task synchronization.

After creating a task in RTOS there is two things that happens first creating the task control block in the stack memory which is a data structure that contains the task ID, priority, status, stack size, context switching counter, and stack pointer. The same configurations as shown in figure (3). Then the created tasks goes to the ready state then check their priority the higher ones goes to the running state until it is finished so it goes to the block state and waits the block times ends to goes again in the ready state and check the priorities of other tasks shown in figure (4). There is a suspended state only in FreeRTOS as where it can be called through a specific API and resumed again through a certain API only.



Figure 55: Task Control Block.

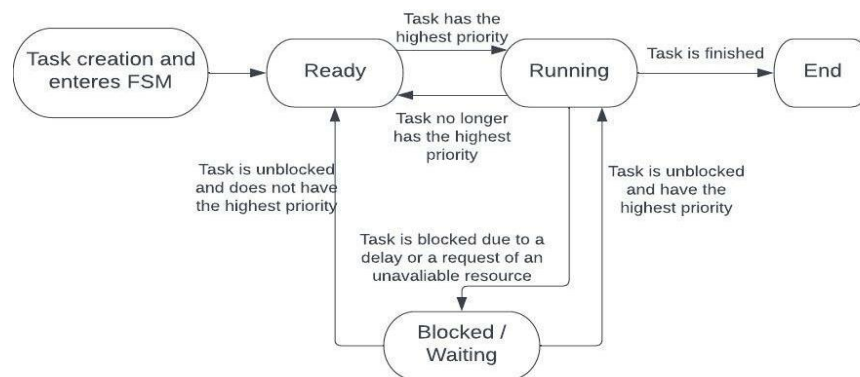


Figure 16: Full Task State Machine.

Scheduling algorithm is the process which chooses and manage another task based on a certain algorithm after removing the running task from the CPU [5, 6, 7, 8]. The tasks were defined based on the deadline, execution time, estimated time of arrival, priority, or resource requirements. Scheduler is activated by events which is brought by interrupts. Scheduling algorithms are divided into two different classifications: preemptive and non-preemptive and both is classified into static and dynamic.

Preemptive means that the task can be interrupted and its waiting or response time is less than that of the non-preemptive. Examples on preemptive: RR, Shortest time first, etc. on the other hand, non-preemptive means that the task cannot be interrupted and its waiting or response time is more than that of the preemptive. Examples of non-preemptive: FCFS, Shortest Job First, etc.

Static scheduling algorithm where all the tasks have the same priority, less complex but unfortunately not as fast as dynamic. But dynamic is a kind of scheduling where the priority are determined while executing the project, faster than static but has a complex hardware.

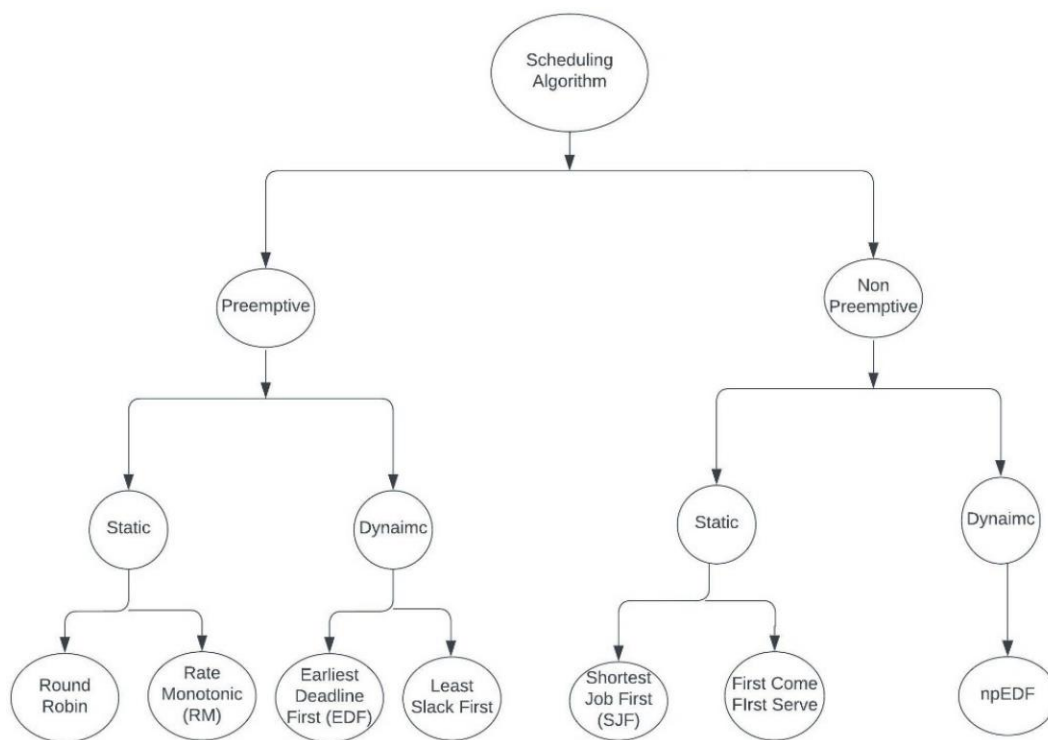


Figure 17: Scheduling Algorithm Classification.

Some of the scheduling algorithms:

- Round Robin: A circular queue implements the RR scheduling method. The selected task is the first in the queue, and the insert is completed at the end of the queue.
- First Come First Serve: non preemptive scheduling algorithms. algorithm gives priority to the queue that is ready and entered first for CPU execution.
- Earliest Deadline First: The EDF is considered a preemptive dynamic scheduling algorithm and considered the most practical algorithm. Highest priority is given to the

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

task with the shortest or nearest deadline. Dynamically assigned and modified priorities are used.

- Priority based: is non-preemptive algorithm which the highest priority is executed first.
- Shortest job first: The SJF is considered a Static priority non preemptive scheduling algorithm. The task with the smallest execution time achieves the highest priority.
- Least Slack Time: The LST is considered a dynamic preemptive scheduling algorithm. The task with the smallest stack time achieves the highest priority.

$$\text{Slack time} = \text{Deadline} - \text{Remaining execution time} - \text{Current time}$$

Deadlock problem occur when two tasks are each unknowingly waiting for a resource “Mutex” that the other task is holding as shown in figure 6. Where the Mutex is a global or shared value most likely be a binary one that can be accessed by other tasks [9].

Criteria of the performance that may be evaluated:

- Task execution time: The total time taken by the task to be fully performed “Worst case execution time”.
- Total execution time: The total execution time of the application.
- Total memory consumption: The Memory size.
- Stack usage and heap memory.
- CPU usage: The percentage of load have been applied to the CPU.
- Interrupt latency: The time between the interrupt request and the and the beginning of execution of the ISR.

Software to measure the performance:

- SEGGER SystemView: has to modes continous recording which is must be connected to J-link debugger and single shot can be conneced to any debugger [10].
- Percepio Tracealyzer [11].

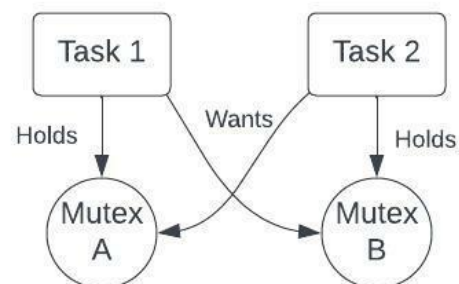


Figure 58: Deadlock Explanation figure.

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

- Simso Simulator [12].
- Keil MDK used for ARM MCs only [13].

Analytical method to measure the performance.

By using the Atmel ICE debugger that works with both AVR and Arm by using Joint Test Action Group (JTAG). The programmer has two available methods by using FreeRTOS APIs “xTaskGetTickCount” and by executing the flow diagram in figure (7).

The other method is by using timer 1 as timer 0 is already taken by RTOS. By initializing the timer1 to work in overflow mode with no prescaling, enabling the interrupt overflow timer1 bit, and enabling the global interrupt and clearing the counting value. And waits for the counter to overflow then the flag will be set and will trigger an interrupt. The counter maximum value is  $0xFFFF = 65,536$ .

And by adding one to global variable whenever this timer OVF interrupt happens and then clear the flag of the interrupt and clear the counter value.

$$T_{cycle} = \frac{1}{Frequency_{operating}}$$

$$time\ wanted = no.\ of\ counter * T_{cycle}$$

## Literature review

The papers i found were mainly divided into 3 parts. First talks about RTOS applications, scheduling algorithms updates and RTOS developments.

In [14] Xing et al. implemented an application for the protection of field workers in polluted sites. The application is a multi-component toxic gas monitoring system based on the STM32 microprocessor, and  $\mu$ OS-II embedded operating system. On the polluted site that is being restored, the system can track the concentration of dangerous gases such hydrogen cyanide, hydrogen Sulphide, carbon monoxide, and Sulphur dioxide in real time and show it on HMI.

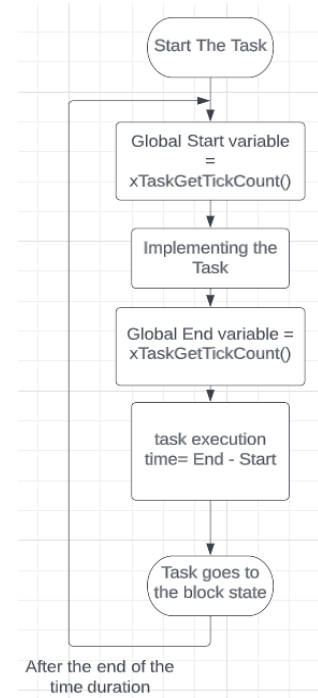


Figure 59: Flow Diagram of the APIs Method to get the task execution time.

The sound light alarm will be activated when the poisonous gas concentration reaches the threshold level to alert the appropriate employees to evacuate quickly.

In [15] Septian, Misbahuddin and Arkan implemented in this research, an air quality monitoring system that measures variables including CO<sub>2</sub>, temperature, humidity, and heat index is constructed using an Arduino Nano and Internet of Things technologies. Display and transmit activities, among others, are run simultaneously using FreeRTOS. CO<sub>2</sub>, temperature, humidity, and heat index are all sensed by MQ135 and DHT22, respectively. ESP8266 Wi-Fi module will be used to periodically send data to a web server using the secure HTTPS POST protocol. A web server is used to establish a secure online application and to receive sensor parameters on the back-end side of the system, allowing users to monitor it remotely.

In [3] Ismael et al. reviewed scheduling algorithms for RTOS, summarize and classifies other papers and compares their different features. The authors talk about concept of scheduling algorithms (Soft, firm, and hard deadlines), components of RTOS (scheduler, fast dispatched latency, memory management, user defined data objects and classes, and function library), properties of RTOS (reliability, stability, scalability, availability, usability, and security) and scheduling criteria (CPU usage, performance and waiting time). The authors talk about different scheduling algorithms and their advantages such as: LST, SJF, FIFO, RR, EDF and FP.

In [6] Putra discussed in this research the priority preemptive scheduling algorithm. With a priority scale of 0 to 10, with 0 being the lowest importance and 10 being the highest. This research discusses two case studies. The result of these two studies is that the average waiting time for priority processes varies. For case 1, the mean turnaround time and the mean waiting time was more than that in case 2. Case 1 is completed more time than in case 2.

In [16] Sinha et al. suggest a new scheduling algorithm called ESRR (Efficient Shortest Remaining Time Round Robin), which combines two preemptive scheduling algorithms called RR and SRTF. In comparison to RR, ESRR decreases total waiting time and turnaround time, as well as waiting times for shorter processes. It also enables longer processes to execute more quickly than SRTF.

In [17] Abu-Dalbouh created a brand-new CPU scheduling method called the mix PI-RR algorithm. The suggested approach decides which jobs should execute and which should wait

to use a mixing of RR and PB scheduling techniques. This innovative technique solves the drawbacks of both scheduling algorithms. The proposed mixed PI-RR algorithm produced better CPU scheduling results, according to the performance metrics and overcoming some issues with the other two algorithms. Also, Zouaoui, Boussaid and Mtibaa in [8] implemented the same approach for scheduling algorithm.

In [18] Unguritu and Nichițelea says this research analyses the characteristics of AUTOSAR (Automotive open system architecture) multicore by an adequate RTOS which coordinating the cores and their tasks in accordance with various strategies. Then the research provides a new and expanded OS with adaptive characteristics that enable each core to manage not only its own activities but also the essential tasks of other cores, but only when necessary. This will enhance the safety perspective of automotive embedded systems. The simulation results show the expected system behavior when taking the suggested task migration technique into consideration. The future work may be tested by using a real multicore system or an emulator.

## Methodology

The application basically is kitchen air control system. The application contains 2 sensors MQ2 gas sensor and LM35. Where at the beginning the sensors have to be initialized and then begin to take their reading after that keep checking the condition of the temperature then if the temperature is higher than 30°C the fan should be on by making the motor rotates clockwise. Then goes to the next condition to make sure that the gas sensor is less than the critical value. If the LPG value is more than 2000 or the smoke value is more than 400 then the fan will work as ventilating fan by moving the motor anti clockwise and display both sensors' readings along with fan state on LCD 20\*4. After that the values should be printed on the lcd and then the loop will repeat continuously from the sensor reading part. Figure (8) shows the whole block diagram of the application. The application is implemented on ATmega32A the frequency used in this application is internal oscillator 1 MHz.



## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

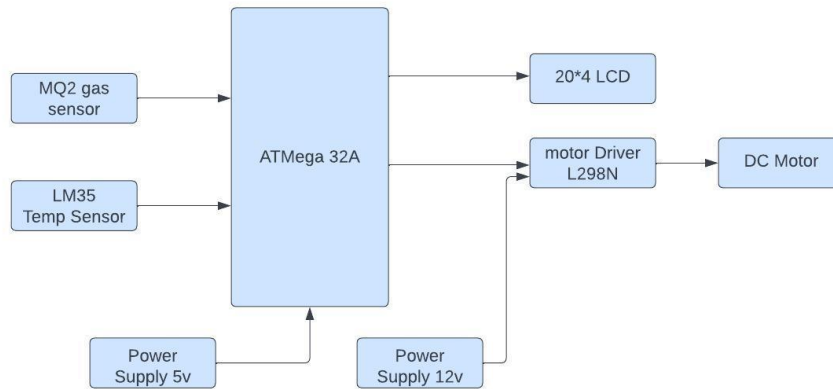


Figure 20: Block diagram of the Implemented Application.

### Experimental or simulation work

The results is checked by two methods software simulations which the author used proteus to ensure the functionality of the desired performance of the application and hardware implementation using Atmel ICE debugger.

The author have tried the simulation in proteus for each code first without RTOS then while using priority based then while using round robin with 3 different time slicing (0.1 sec, 10m sec, 100μ sec) and by using priority based round robin. Where the LEDs is just for illustration when the task begin to execute until the task finishes execution and then shows the on and off states and the execution on the oscilloscope.

Figure 9 shows the simulation in proteus and figure 10,11 shows some values while using the debugger.

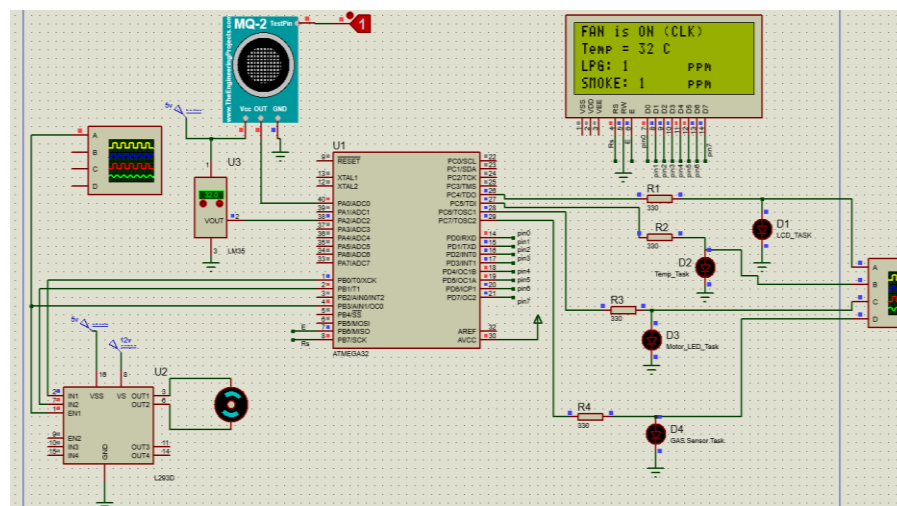


Figure 61: Proteus Simulation while the Temperature more than 30.

# Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

*(Lcd_Handler)	{struct tskTaskControlBlock{data}@0x017b	struct tskTaskControlBlock{data}@0x01a6
pxTopOfStack	0x017b	StackType_t*(data)@0x01a6
xStateListItem	(ListItem_t{data}@0x01a8)	ListItem_t{data}@0x01a8
xEventListItem	(ListItem_t{data}@0x01b2)	ListItem_t{data}@0x01b2
uxPriority	1	UBaseType_t{data}@0x01bc
pxStack	0x013e	StackType_t*(data)@0x01bd
pcTaskName	0x01bf	char[12]{data}@0x01bf
uxTCBNumber	1	UBaseType_t{data}@0x01cb
uxTaskNumber	0	UBaseType_t{data}@0x01cc
ulNotifiedValue	0x01cd	uint32_t[1]{data}@0x01cd
ucNotifyState	0x01d1	uint8_t[1]{data}@0x01d1

Figure 22: Example of the Created TCB.

## Main results

The best scheduling algorithm for this application is priority based round robin with total execution time is 0.065 sec that shows perfect execution time for the total application and each task individually and the worst execution time was by using RR with time slicing 100  $\mu$ sec which was 1.35 and the application without using RTOS execution time was 4.58 sec.

The memory consumption of the application without RTOS was RAM of 8% full, and ROM of 18.7% full while the PBRR memory consumption was RAM of 89.9% full, of 31.5% full

Figure 12 shows a simulation of the task execution made by using Simso for PB scheduling algorithm only. Also, Simso shows CPU usage of 31.734 %.

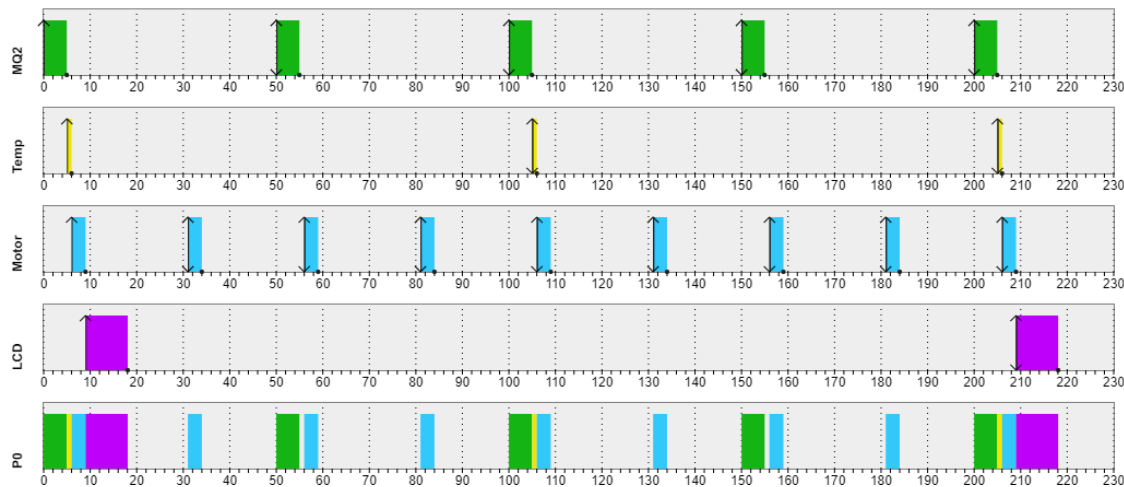


Figure 63: Simulation Shows the Task Execution Based on PB Scheduling Algorithm.

## Conclusion

In conclusion this research paper talks briefly about RTOS operation and how it works and its types. This paper made a good comparison between scheduling algorithms and examples

## Implementing and Measuring the Performance of PB, RR and PBRR Scheduling Algorithms on ATmega32A using FreeRTOS

for each type. The author made an shed light on the increment of death and injuries because of LPG burns and leakages. So, the author made an application by using FreeRTOS with different scheduling algorithms PB, RR “ with 3 different time slicing “1/10, 1/100, 1/10000” and PBRR that is implemented on ATmega32A by using FreeRTOS. The author shows the PPM values of CO and the time before critical symptoms begins.

The simulations shows the best scheduling algorithm in time execution which was PBRR with time = 0.0656 sec while the other algorithms takes more time than that and without RTOS takes 4.581 sec. however, the memory consumption of the application without RTOS was 8% full, ROM was 18.7% while the PBRR memory consumption was 89.9% full, ROM was 31.5% full. In the future the author will try to implement these scheduling algorithms with different RTOS and on a multicore MC.

## References

- [1] R. Jin, P. Wu, J. K. Ho, X. Wang and C. Han, "Five-year epidemiology of liquefied petroleum gas Erelated burns," *BURNS*, vol. 44, pp. 210-217, 2018.
- [2] "Hazardous Substance Fact Sheet (Liquefied Petroleum Gas)," NJ Health, New Jersey, 2010.
- [3] G. A. Ismael, A. A. Salih, A. AL-Zebari, N. Omar, K. J. Merceedi, A. J. Ahmed, N. O. M. Salim, S. S. Hasan, S. F. Kak, . I. . M. Ibrahim and H. M. Yasin, "Scheduling Algorithms Implementation for Real Scheduling Algorithms Implementation for Real," *Asian Journal of Research in Computer Science*, pp. 35-51, 2021.
- [4] T. Nguyen, B. Anh and S.-L. Tan, "REAL-TIME OPERATING SYSTEMS FOR SMALL MICROCONTROLLERS," 2009.
- [5] J. Alhiyafi, "PERFORMANCE EVALUATION OF NON-PREEMPTIVE EARLIEST DEADLINE FIRST SCHEDULING TECHNIQUES," *ICIC EXPRESS LETTERS*, vol. 10, pp. 17-24, 2019.
- [6] T. D. Putra, "Analysis of Priority Preemptive Scheduling Algorithm: Case Study," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 11, no. 1, 2022.
- [7] T. D. Putra, "Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 9, no. 4, 2020.

- [8] S. Zouaoui, L. Boussaid and A. Mtibaa, "Priority based round robin (PBRR) CPU scheduling algorithm," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 190-202, 2019.
- [9] M. Begum, O. Faruque, M. W. R. Miah and B. C. Das, "An Improved Safety Detection Algorithm Towards Deadlock Avoidance," in *IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Malaysia, 2020.
- [10] "SEGGER System View," SEGGER, [Online]. Available: <https://www.segger.com/products/development-tools/systemview/technology/what-is-systemview/>.
- [11] "Tracealyzer Visual Runtime Insights," Percepio Sensing software, [Online]. Available: <https://percepio.com/tracealyzer/>.
- [12] "SimSo documentation," SimSo, [Online]. Available: <https://projects.laas.fr/simso/doc/introduction.html#what-is-simso>.
- [13] "Keil MDK," armDeveloper, [Online]. Available: <https://developer.arm.com/Tools%20and%20Software/Keil%20MDK>.
- [14] Y. Xing, J. Zhu, Y. Fu, Y. Zhang and Q. Qiao, "Embedded Toxic Gas Monitor Based on MCOS-II," in *International Conference on Mechatronics and Automation*, 2019.
- [15] B. Septian, M. Misbahuddin and F. Arkan, "FREERTOS BASED AIR QUALITY MONITORING SYSTEM USING SECURE INTERNET OF THINGS," *Jurnal Teknik Informatika (JUTIF)*, vol. III, no. 1, pp. 147-153, 2022.
- [16] P. Sinha, B. Prabadevi, S. Dutta, D. N and N. Kumari, "Efficient Process Scheduling Algorithm using RR and SRTF," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Vellore, India, 2020.
- [17] H. M. Abu-Dalbouh, "A New Combination Approach to CPU Scheduling based on Priority and Round-Robin Algorithms for Assigning a Priority to a Process and Eliminating Starvation," (*IJACSA*) *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, pp. 541-546, 2022.
- [18] M. G. Unguritu and T. C. Nichiștea, "Adaptive Real-Time Operating System in Automotive Multicore Embedded Systems," in *International Conference on System Theory, Control and Computing (ICSTCC)*, Iasi, 2021.