



SPRINTS

# Project EDF SCHEDULER

Version 1.0

Prepared by:

**Bits**  **Tribe**

Hazem Ashraf  
Mohamed Abdelsalam  
Abdelrhman Walaa  
Amr ElAbd

**July 2023**



## Table Of Content

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Incorporating Missed Thesis Changes.....</b>	<b>4</b>
2.1. EDF Scheduler's Idle Task.....	4
2.2. EDF Scheduler's xTaskIncrementTick Function.....	4
2.2.1. Calculating New Task Deadline.....	4
2.2.1. Modifying Preemption Method.....	5
<b>3. System Design.....</b>	<b>6</b>
3.1. Task Set Parameters.....	6
<b>4. System Analysis.....</b>	<b>7</b>
4.1. Analytical Methods.....	7
4.1.1. System Hyperperiod Calculation.....	7
4.1.2. System CPU Load Calculation.....	7
4.1.3. System Schedulability Check using URM.....	8
4.1.4. System Schedulability Check using Time-Demand Analysis Techniques.....	8
4.1.5. System Analysis with GPIOs and Logic Analyzer.....	10
4.2. System Simulation.....	12
4.2.1. Offline Simulation.....	12
4.3. Run-Time Analysis Method.....	15
4.3.1. System Analysis with Trace Macros and Logic Analyzer.....	15
4.3.1.1. Description.....	16
4.3.1.2. Analysis.....	16
4.3.2. System CPU Load.....	19
4.3.2.1. Description.....	19
4.3.2.2. Analysis.....	19
4.3.3. Run-Time Analysis Method Conclusion.....	20
<b>5. References.....</b>	<b>21</b>



## Report on EDF Scheduler

### 1. Introduction

The goal of this project is to implement a scheduling algorithm called Earliest Deadline First (EDF) for a real-time system. **EDF** is a dynamic priority-based preemptive scheduling policy, which means that the priority of a task can change during its execution, and any task can be interrupted by a lower *Periodicity* task. This scheduling algorithm ensures that tasks with closer deadlines are given higher priority, allowing for efficient task management in real-time systems.

In this project, we have identified six tasks to be scheduled using the EDF algorithm. The first four tasks are *Button\_1\_Monitor*, *Button\_2\_Monitor*, *Periodic\_Transmitter*, and *Uart\_Receiver*, each with their specific periodicity and deadlines. *Button\_1\_Monitor* and *Button\_2\_Monitor* tasks monitor the rising and falling edges of buttons 1 and 2, respectively, and send events to the consumer task. *Periodic\_Transmitter* sends a periodic string every 100ms to the consumer task, while *Uart\_Receiver* receives strings from other tasks and writes them to the UART.

To simulate a heavier load, two additional tasks, *Load\_1\_Simulation* and *Load\_2\_Simulation*, have been introduced. These tasks are implemented as empty loops that need to be executed a specific number of times to achieve the desired execution time. By adjusting the loop iterations of these tasks, we can simulate a workload that will impact the overall system performance.

Overall, this project aims to showcase the implementation of the **EDF** scheduling algorithm and simulate a real-time system with different tasks, including the addition of heavier loads. By properly prioritizing and scheduling tasks based on their deadlines, we can ensure the efficient execution of critical tasks while maintaining system responsiveness and meeting the specified timing requirements.



## 2. Incorporating Missed Thesis Changes

This section highlights the additional changes that were incorporated into the **EDF Scheduler**, which were not originally mentioned in the thesis. By addressing the omissions from the thesis, the EDF scheduler has been further refined and improved to better meet the requirements of the system.

These changes specifically pertain to the “*tasks.c*” file. For further information and specific references, please refer to the *References* section of the document.

### 2.1. EDF Scheduler's Idle Task

In the provided code snippet, a modification has been made to the idle task, specifically in the “**prvIdleTask**” function, to ensure that its *deadline* remains the *furthest* among all tasks.

```

3646         if ( !listLIST_IS_EMPTY(&xReadyTasksListEDF) )
3647         {
3648             listSET_LIST_ITEM_VALUE( &( ( pxCurrentTCB )->xStateListItem ), pxCurrentTCB->xTaskPeriod + xTaskGetTickCount() );
3649         }
3650         else
3651         {
3652             /* Do Nothing */
3653         }
3654     }
3655 #endif

```

The modification aims to prioritize tasks with closer deadlines over the **Idle Task** by incrementing the idle task's period and setting a new deadline in the ready list. This ensures that the idle task always has the farthest deadline among all tasks.

Consequently, the idle task remains idle only when there are no pending tasks, and once a task is added to the ready list, the idle task's period is incremented to yield the CPU to the task with the closest deadline.

### 2.2. EDF Scheduler's xTaskIncrementTick Function

#### 2.2.1. Calculating New Task Deadline

In the “**xTaskIncrementTick**” function, during each tick increment, the EDF scheduler calculates the new deadline for each task and inserts it into the appropriate position in the EDF ready list.

This ensures that the tasks are prioritized based on their deadlines, allowing for effective task scheduling and adherence to timing constraints in the real-time system.



### 2.2.1. Modifying Preemption Method

In the “**xTaskIncrementTick**” function, another modification is made to the preemption mechanism to ensure that a context switch occurs as soon as a new task becomes available in the EDF ready list.

Instead of considering priority alone, the preemption is now based on comparing the periodic time of the task in the ready list to the periodic time of the currently running task. This means that a task with a sooner deadline will preempt a task with a larger deadline, prioritizing timely execution and meeting timing constraints in the system.

```

2934  □      #if ( configUSE_PREEMPTION == 1 )
2935  □      {
2936  □          #if (configUSE_EDF_SCHEDULER == 1)
2937  □          {
2938  □              /*
2939  □              compare the periodic time of the task in the ready list
2940  □              to the periodic time of the current running task on CPU
2941  □              */
2942  □              if( pxTCB->xTaskPeriod <= pxCurrentTCB->xTaskPeriod )
2943  □              {
2944  □                  /*set contextswitch flag to true*/
2945  □                  xSwitchRequired = pdTRUE;
2946  □              }
2947  □              else
2948  □              {
2949  □                  mtCOVERAGE_TEST_MARKER();
2950  □              }
2951  □          }
2952  □      #else
2953  □      {
2954  □          if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
2955  □          {
2956  □              xSwitchRequired = pdTRUE;
2957  □          }
2958  □          else
2959  □          {
2960  □              mtCOVERAGE_TEST_MARKER();
2961  □          }
2962  □      }
2963  □      #endif /*configUSE_EDF_SCHEDULER*/
2964  □  }
2965  □  #endif /* configUSE_PREEMPTION */

```



### 3. System Design

#### 3.1. Task Set Parameters

Task ID	Task	Task Type	Periodicity (ms)	Execution Time (ms)	Deadline (ms)
T1	Button_1_Monitor	Periodic	50	0.0013	50
T2	Button_2_Monitor	Periodic	50	0.0013	50
T3	Periodic_Transmitter	Periodic	100	0.0055	100
T4	Uart_Receiver	Periodic	20	0.0127	20
T5	Load_1_Simulation	Periodic	10	5	10
T6	Load_2_Simulation	Periodic	100	12	100

**Note:** The execution time of tasks T1 ([Button\\_1\\_Monitor](#)), T2 ([Button\\_2\\_Monitor](#)), T3 ([Periodic\\_Transmitter](#)), and T4 ([Uart\\_Receiver](#)) is determined based on the actual execution time measured using GPIOs and the logic analyzer.



## 4. System Analysis

### 4.1. Analytical Methods

#### 4.1.1. System Hyperperiod Calculation

The system **Hyperperiod** is calculated by finding the *Least Common Multiple (LCM)* of all the tasks' *Periodicities*, representing the smallest multiple where all tasks complete a full cycle.

To calculate Hyperperiod = LCM (All Tasks' Periodicities)  
 = LCM (10, 20, 50, 100) = **100 ms**

#### 4.1.2. System CPU Load Calculation

The **CPU Load** is calculated by dividing the total *Execution Time* of tasks (**R**) by the total available time (**C**), resulting in the utilization of the CPU. **Utilization = R/C**.

where, **R**: Requirements which in simple terms is the BUSY TIME

**C**: Capacity which in simple terms is BUSY TIME + IDLE TIME

BUSY TIME = Execution time \* (Hyperperiod / Periodicity)

Task ID	Periodicity (ms)	Execution Time (ms)	Busy Time
T1	50	0.0013	$(100/50)*0.0013 = 0.0026$
T2	50	0.0013	$(100/50)*0.0013 = 0.0026$
T3	100	0.0055	$(100/100)*0.0055 = 0.0055$
T4	20	0.0127	$(100/20)*0.0127 = 0.0635$
T5	10	5	$(100/10)*5 = 50$
T6	100	12	$(100/100)*12 = 12$
SUM			<b>62.0742</b>

To calculate CPU Load = Busy Time / Total Time  
 = 62.07 / 100 = **0.6207 = 62.07%**



#### 4.1.3. System Schedulability Check using URM

**Rate Monotonic Utilization** is a method used to calculate the maximum possible CPU utilization in a real-time system by summing up the utilization factors of individual tasks.

$$U = \sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

where, **U** = Total Utilization  
**C** = Execution time  
**P** = Periodicity  
**N** = Number of tasks

**To calculate Urm** (Rate Monotonic Utilization) =  $6 ( 2 ^ { 1 / 6 } ) - 1 = 0.734$

With a **CPU Load** of **62.07%** and **Urm** of **73.4%**, the system is **Schedulable** as the CPU Load is lower than the Urm. This indicates that the tasks can be scheduled within their timing constraints and the system can meet its real-time requirements.

#### 4.1.4. System Schedulability Check using Time-Demand Analysis Techniques

The **Worst Response Time** is a method used to determine the maximum time it takes for a task to complete its execution and provide a response. It helps in analyzing the worst-case scenario for task response times in a real-time system.

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k \quad \text{for } 0 < t \leq p_i$$

where, **W** = Worst response time  
**E** = Execution time  
**P** = Periodicity  
**T** = Time instance





Task ID	t	w(t)			
<b>T1</b>	1	5			
	10	5	w(10) < deadline	2.5 < 10	<b>T1 is Schedulable</b>
<b>T2</b>	1	0.5127			
	5	2.5127			
	15	7.5127			
	20	<b>10.0127</b>	w(20) < deadline	10.0127 < 20	<b>T2 is Schedulable</b>
<b>T3</b> <b>T4</b>	1	0.501			
	10	5.01			
	20	10.01			
	30	15.02			
	40	20.26			
	50	<b>25.03</b>	w(50) < deadline	25.033 < 50	<b>T3 is Schedulable</b>
<b>T5</b>	1	5.5			
	20	15.01			
	40	25.02			
	60	35.04			
	80	45.05			
	100	<b>55.06</b>	w(100) < deadline	55.069 < 100	<b>T5 is Schedulable</b>
<b>T6</b>	1	12.5			
	20	23.01			
	40	34.02			
	60	45.04			
	80	56.05			
	100	<b>67.06</b>	w(100) < deadline	67.069 < 100	<b>T6 is Schedulable</b>

Tasks **T1**, **T2**, **T3**, **T4**, **T5**, and **T6** are considered **Schedulable** as they successfully meet their respective deadlines



#### 4.1.5. System Analysis with GPIOs and Logic Analyzer

**Logic Analyzers** are valuable tools for capturing and displaying these signals, allowing for detailed analysis. Setting and clearing GPIO pins when specific events occur can visually indicate system behavior.

Here are the results obtained from the Logic Analyzer.

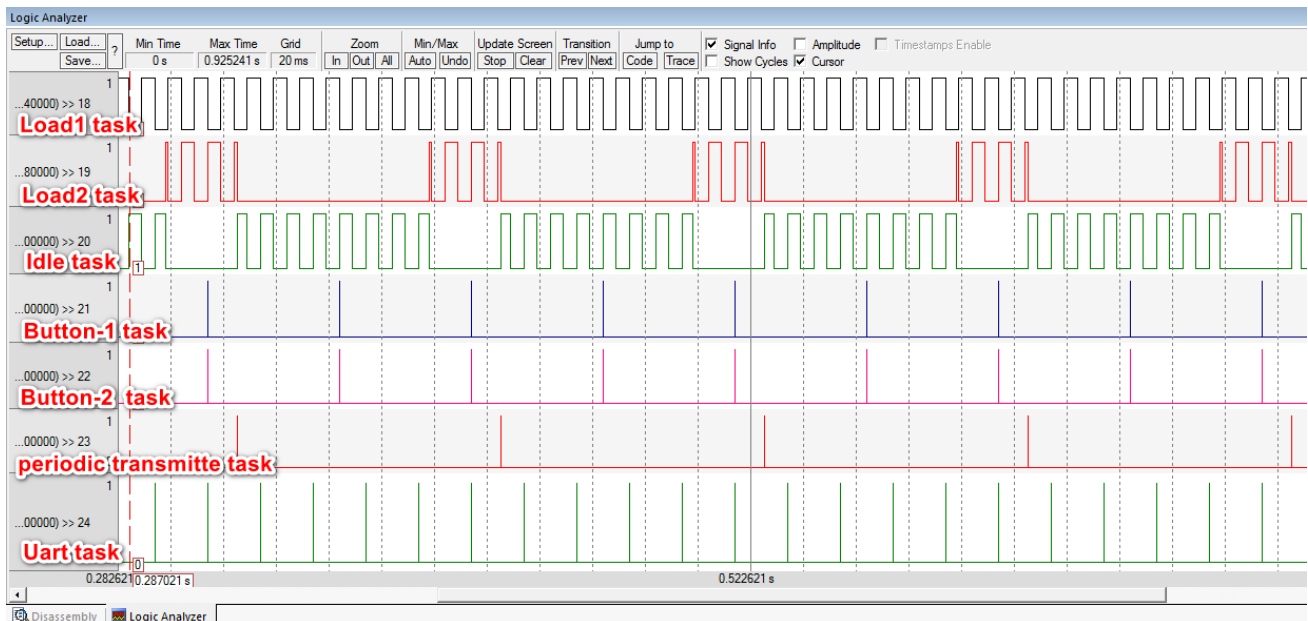


Figure 1. System Timeline using GPIO Pins

1. Tasks “Button\_1\_Monitor” and “Button\_2\_Monitor” execution on the Logic Analyzer:

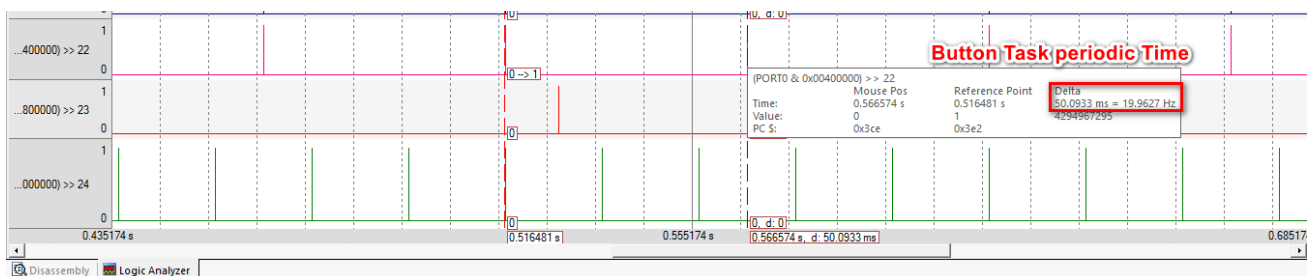


Figure 2. Tasks “Button\_1\_Monitor” and “Button\_2\_Monitor” Periodicities

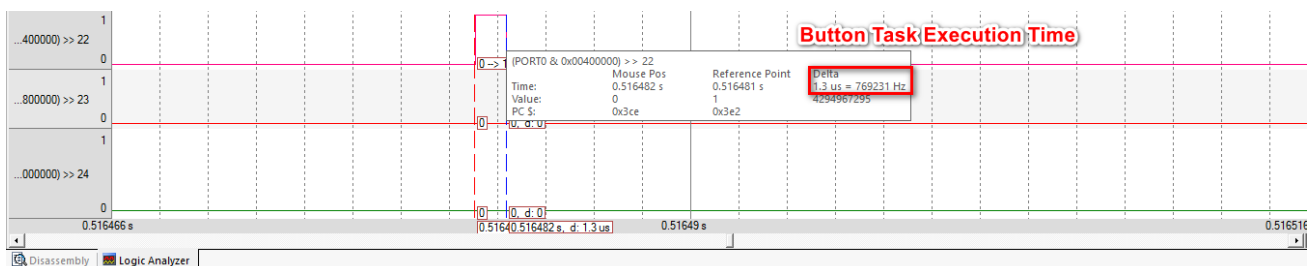


Figure 3. Tasks “Button\_1\_Monitor” and “Button\_2\_Monitor” Execution Times



## 2. Task “Periodic\_Transmitter” execution on the Logic Analyzer:



Figure 4. Task “Periodic\_Transmitter” Periodicity

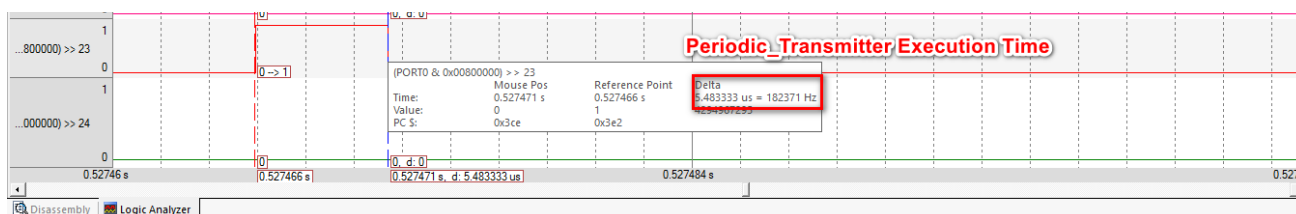


Figure 5. Task “Periodic\_Transmitter” Execution Time

## 3. Task “Uart\_Receiver” execution on the Logic Analyzer:

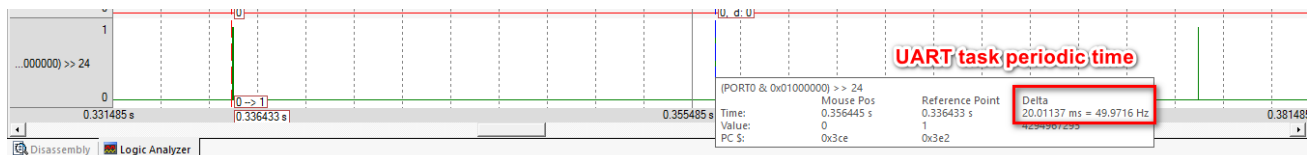


Figure 6. Task “Uart\_Receiver” Periodicity

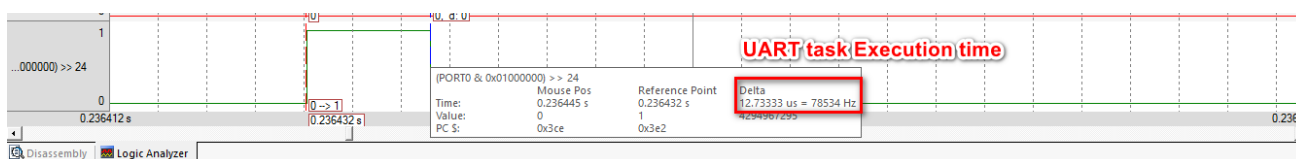


Figure 7. Task “Uart\_Receiver” Execution Time



#### 4. Tasks “Load\_1\_Simulation” and “Load\_2\_Simulation” execution on the Logic Analyzer:

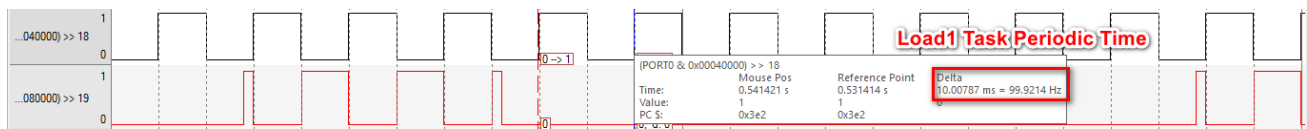


Figure 8. Task “Load\_1\_Simulation” Periodicity

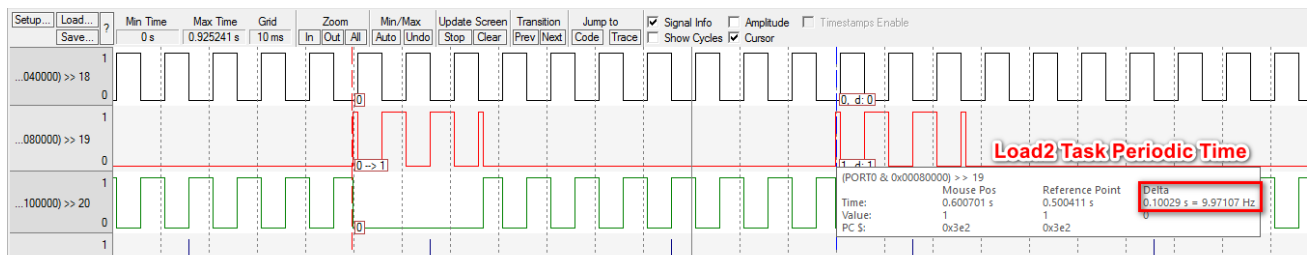


Figure 9. Task “Load\_2\_Simulation” Periodicity

## 4.2. System Simulation

### 4.2.1. Offline Simulation

The *SimSo Real-Time Scheduling Simulator* demonstrates that the system is schedulable. The calculated **CPU Load** of **0.620** aligns closely with the manual calculations. Additionally, the system exhibits schedulability as all tasks do successfully meet its **deadline**.

	Total load	Payload	System load
CPU 1	0.6204	0.6204	0.0000
Average	0.6204	0.6204	0.0000

Figure 10. CPU Load using SimSo Simulator



All tasks successfully meet their deadlines, indicating that they complete their execution within the designated time frames.

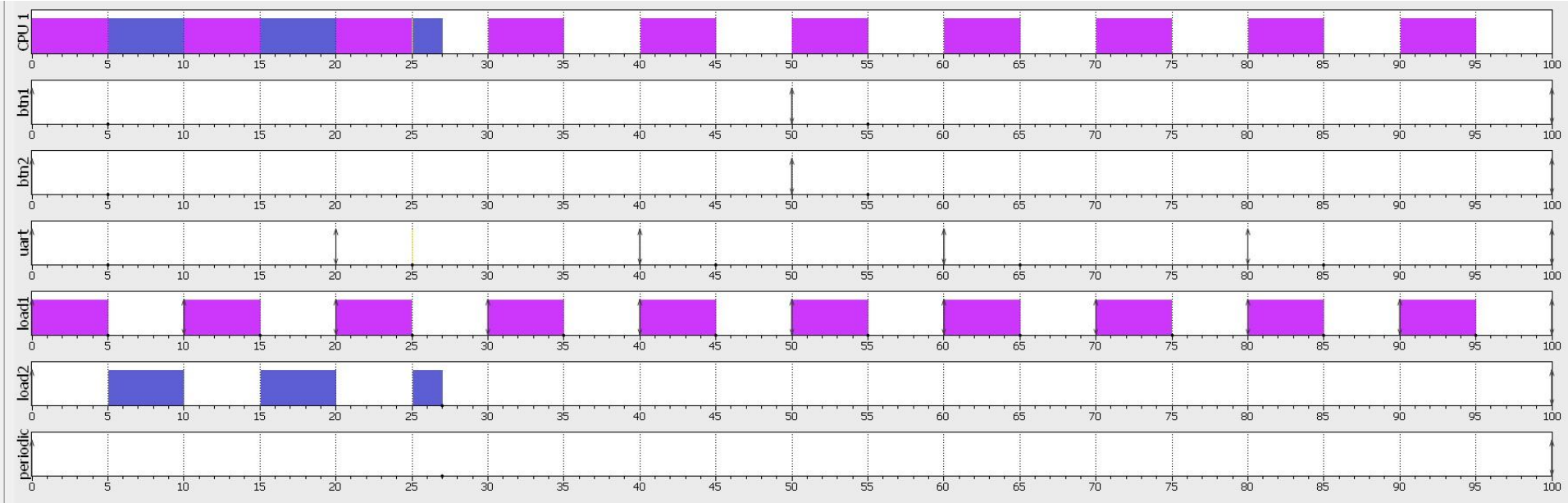


Figure 11. System Timeline using SimSo Simulator



Date (cycles)	Date (ms)	Message
50000000	50.0	btn1_2 Activated.
50000000	50.0	btn2_2 Activated.
50000000	50.0	load1_6 Activated.
50000000	50.0	load1_6 Executing on CPU 1
55000000	55.0	load1_6 Terminated.
55000000	55.0	btn1_2 Executing on CPU 1
55001300	55.0013	btn1_2 Terminated.
55001300	55.0013	btn2_2 Executing on CPU 1
55002600	55.0026	btn2_2 Terminated.

**button 1 & button 2 are activated at period 50**

Figure 12. System Log of Tasks “*Button\_1\_Monitor*” and “*Button\_2\_Monitor*” on SimSo Simulator

Date (cycles)	Date (ms)	Message
10000000	10.0	load1_2 Activated.
10000000	10.0	load2_1 Preempted! ret: 7008600
10000000	10.0	load1_2 Executing on CPU 1
15000000	15.0	load1_2 Terminated.
15000000	15.0	load2_1 Executing on CPU 1
20000000	20.0	uart_2 Activated.
20000000	20.0	load1_3 Activated.
20000000	20.0	load2_1 Preempted! ret: 2008600
20000000	20.0	load1_3 Executing on CPU 1
25000000	25.0	load1_3 Terminated.

**Load1 is activated at period = 10 and finish  
executing after 5ms**

**UART is activated at period = 20**

Figure 13. System Log of Tasks “*Uart\_Receiver*” and “*Load\_1\_Simulation*” on SimSo Simulator

Date (cycles)	Date (ms)	Message
90000000	90.0	load1_10 Executing on CPU 1
95000000	95.0	load1_10 Terminated.
100000000	100.0	load2_2 Activated.
100000000	100.0	periodic_2 Activated.
100000000	100.0	btn1_3 Activated.
100000000	100.0	btn2_3 Activated.
100000000	100.0	uart_6 Activated.
100000000	100.0	load1_11 Activated.
100000000	100.0	load1_11 Executing on CPU 1

**Load2 and Periodic tasks are activated at period = 100**

Figure 14. System Log of Tasks “*Periodic\_Transmitter*” and “*Load\_2\_Simulation*” on SimSo Simulator



### 4.3. Run-Time Analysis Method

This method allows us to measure and visualize the execution of tasks, tick, and the idle task, and determine the CPU usage time using timer 1 and **Trace Macros** in the Keil simulator. The **Logic Analyzer**, along with GPIOs, provides a graphical representation of the task execution timeline, aiding in the analysis and verification of the scheduling algorithm's effectiveness.

#### 4.3.1. System Analysis with Trace Macros and Logic Analyzer

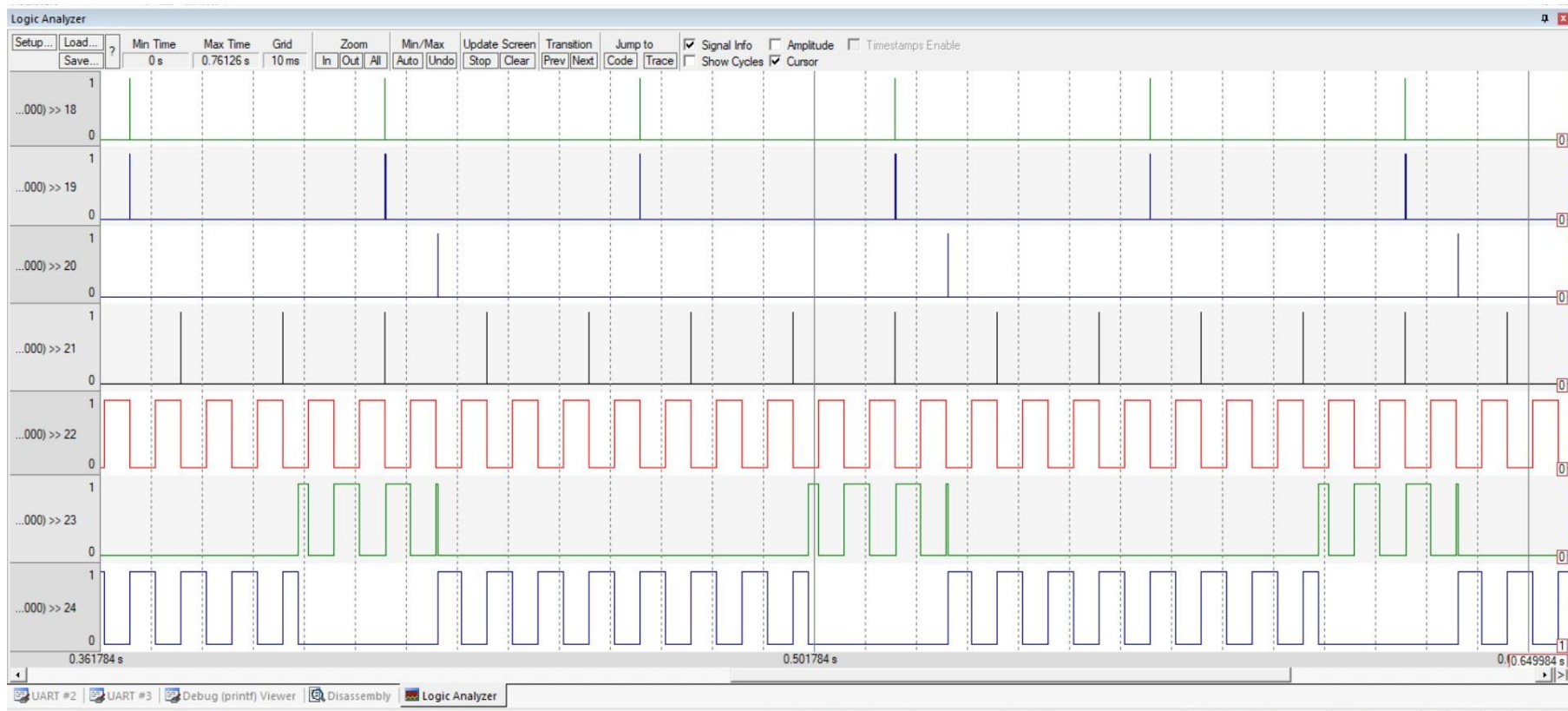


Figure 15. System Timeline using Trace Macros



#### 4.3.1.1. Description

Analyzing the pin signals on the **Logic Analyzer** will provide valuable insights into the timing and sequence of task execution, allowing for the verification of the scheduling algorithm and the detection of any overlaps or delays.

Additionally, the pin representing the "**Idle**" Task will help identify periods of system inactivity when no tasks are being executed.

In *Figure 3*, the pins on the Logic Analyzer are configured as follows:

- Pin 3 (18): Represents the "**Button\_1\_Monitor**" Task.
- Pin 4 (19): Represents the "**Button\_2\_Monitor**" Task.
- Pin 5 (20): Represents the "**Periodic\_Transmitter**" Task.
- Pin 6 (21): Represents the "**Uart\_Receiver**" Task.
- Pin 7 (22): Represents the "**Load\_1\_Simulation**" Task.
- Pin 8 (23): Represents the "**Load\_2\_Simulation**" Task.
- Pin 9 (24): Represents the "**Idle**" Task.

#### 4.3.1.2. Analysis

During the execution of the Keil simulator with trace macros and GPIOs, the Earliest Deadline First (EDF) scheduler operates as expected. The timeline of each task is uniform, which indicates that the tasks are meeting their respective deadlines. This demonstrates the effectiveness of the EDF scheduling algorithm in ensuring timely task completion.

To validate these observations, we conducted the simulation using tasks that have different execution times, including Load\_1\_Simulation and Load\_2\_Simulation. By analyzing the task execution timelines and closely observing the task switches, we confirmed that the EDF scheduler effectively schedules and prioritizes tasks based on their deadlines.





- Task “[Load\\_1\\_Simulation](#)” execution on the logic analyzer:

Its **Periodicity** is **10 ms** (absolute) using `xTaskDelayUntil` API.

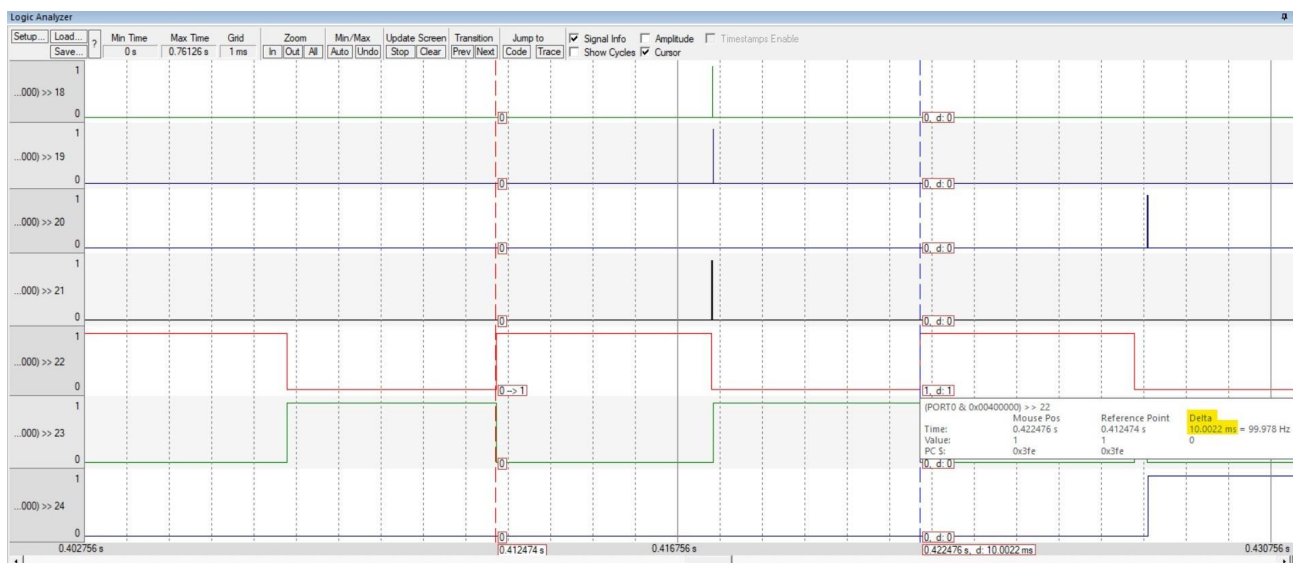


Figure 16. Task “[Load\\_1\\_Simulation](#)” Periodicity

Its **Execution Time** is **5 ms** as required.

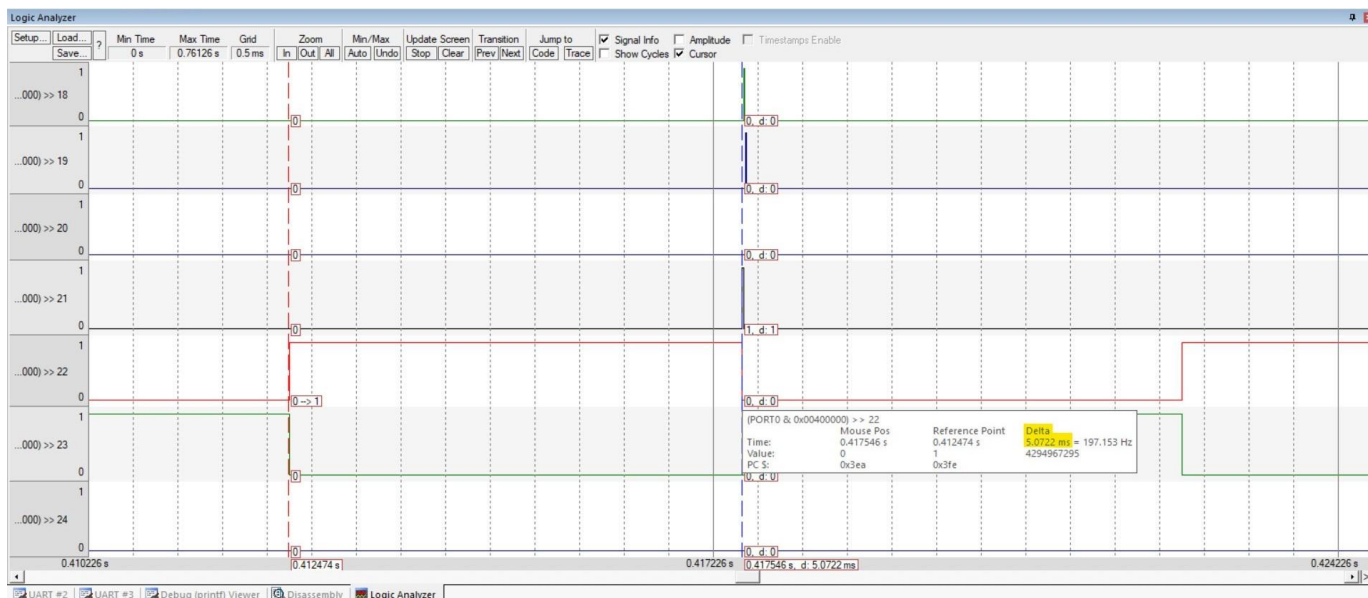


Figure 17. Task “[Load\\_1\\_Simulation](#)” Execution Time



## 2. Task “Load\_2\_Simulation” execution on the logic analyzer:

Its **Periodicity** is **100 ms** (absolute) using `xTaskDelayUntil` API.

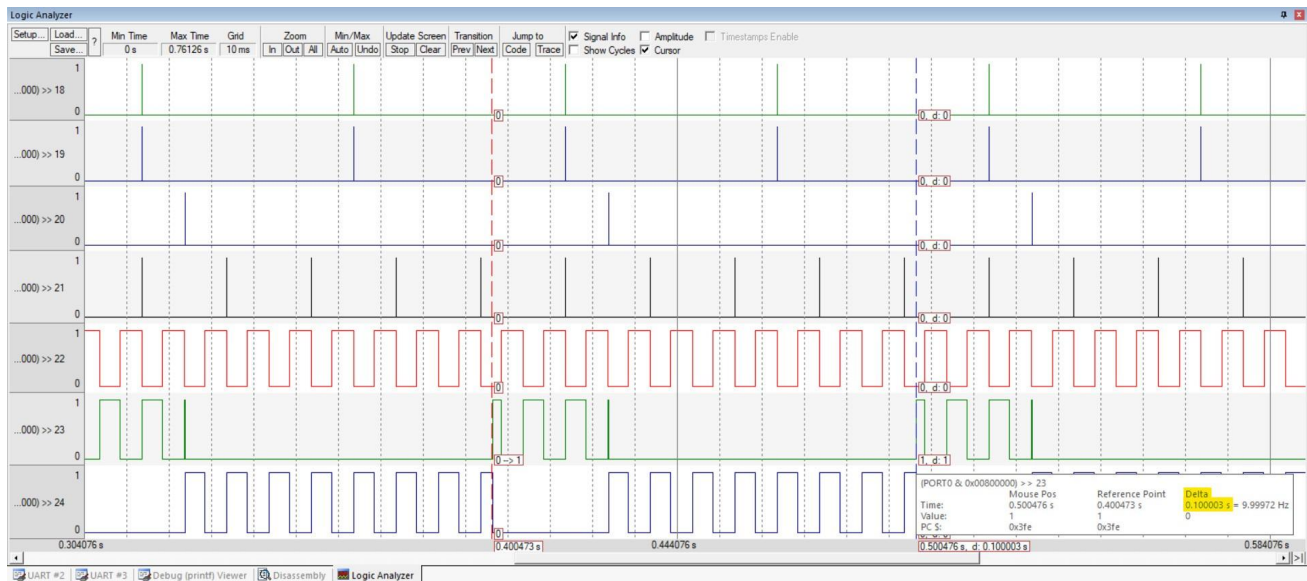


Figure 18. Task “Load\_2\_Simulation” Periodicity

Its **Execution Time** is **27 ms** ( i.e. The logic analyzer captures the execution of the “Load\_1\_Simulation” task, showing a preemption of **15 ms** occurring *three* times and the remaining **12 ms** as required. )

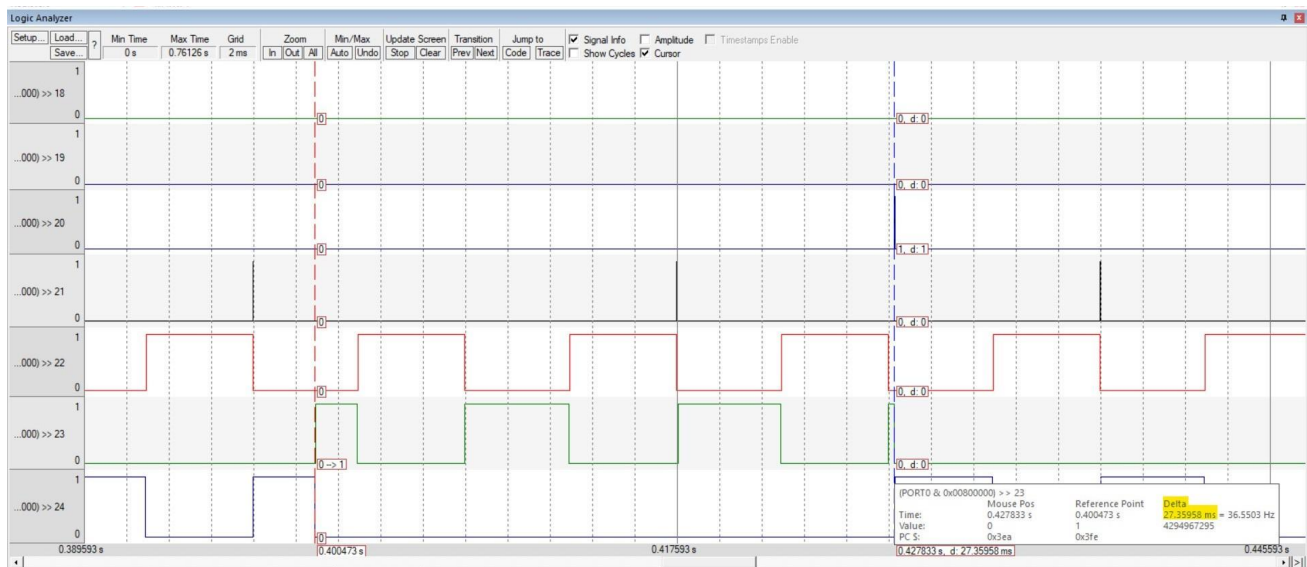


Figure 19. Task “Load\_2\_Simulation” Execution Time



### 4.3.2. System CPU Load


Watch 1		
Name	Value	Type
 Cpu_Load	62.3051147	float

Figure 20. CPU Load using Trace Macros

#### 4.3.2.1. Description

The following equation was utilized to calculate the **CPU Utilization** and assess the effectiveness of the scheduling algorithm.

$$\text{"CPU load} = (\text{Total execution time of all tasks} / \text{Current System Time}) * 100"$$

The objective was to compare the results obtained from trace macros in the EDF scheduler with those derived from analytical methods and the Simso tool, ensuring consistency in the measurements and evaluating the efficiency of the scheduling algorithm.

#### 4.3.2.2. Analysis

For the analysis, the **Trace Macros** were employed to measure the total execution time of all tasks, providing precise tracking of each task's time utilization during system execution. Concurrently, the current system time was recorded to calculate the **CPU Load**, which is equal to **62.3%**.



#### 4.3.3. Run-Time Analysis Method Conclusion

The usage of **Trace Macros** to get the **CPU Load** gave results **equivalent** to that of the analytical methods and Simso results . Based on the analysis performed in this report, it can be concluded that the EDF scheduler implemented in the freeRTOS system functions correctly.

The comparison between task execution order and their respective deadlines demonstrated that the scheduler correctly prioritized tasks based on their deadlines. Tasks with earlier deadlines were consistently executed before those with later deadlines, as required by the EDF algorithm. The absence of missed deadlines or delayed task completions suggested that the scheduler successfully managed task scheduling in a real-time environment.

Therefore, this report confirms the functionality of the implemented **EDF Scheduler** in *FreeRTOS*, validating its ability to correctly prioritize tasks based on deadlines and ensure timely task completion.



## 5. References

- **Carraro, E.** (2016). *Implementation and Test of EDF and LLREF Schedulers in FreeRTOS* (Unpublished master's thesis). [University of London and the Diploma of Imperial College].