EDF Scheduler

Prepared by
Team 1

Ahmed Hesham
Alaa Hisham
Hossam Elwahsh
Sarah Mohamed
Khaled Mustafa

Sprints

VERIFYING EDF SCHEDULER

1. INTRODUCTION

Earliest Deadline First (EDF) is a scheduling algorithm that adopts a dynamic priority-based preemptive scheduling policy, meaning that the priority of a task can change during its execution, and the processing of any task is interrupted by a request for any higher priority task.

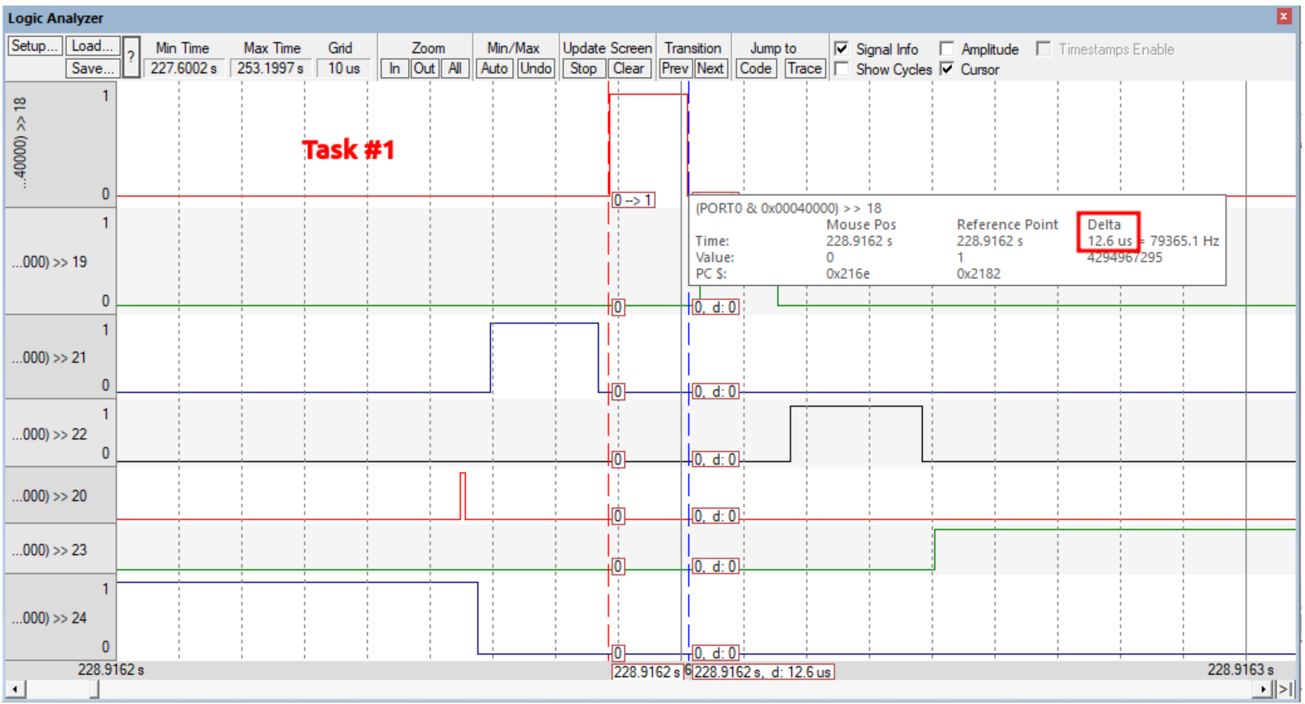
In this project, we are designing an EDF scheduler based on FreeRTOS.

2. TASKS

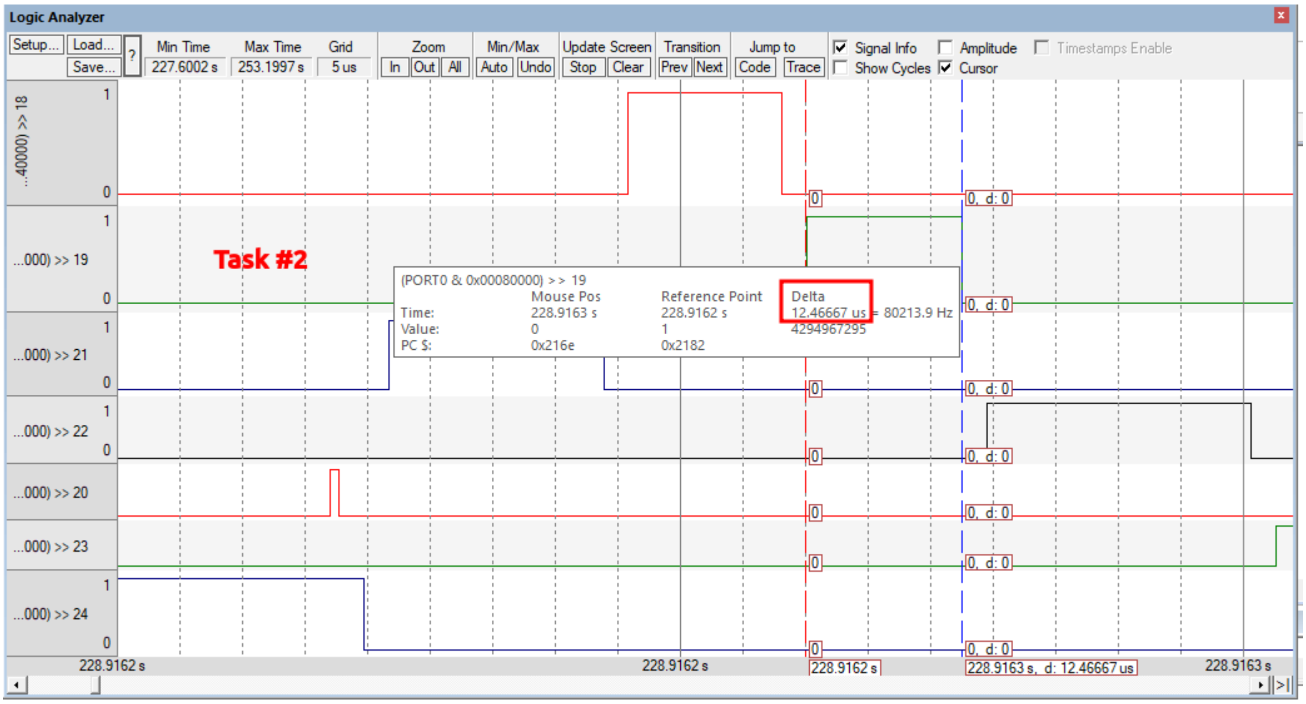
2.1. Introduction

You will find here the parameters for each task used, regarding the execution time of each task, you can find them in the next screenshots.

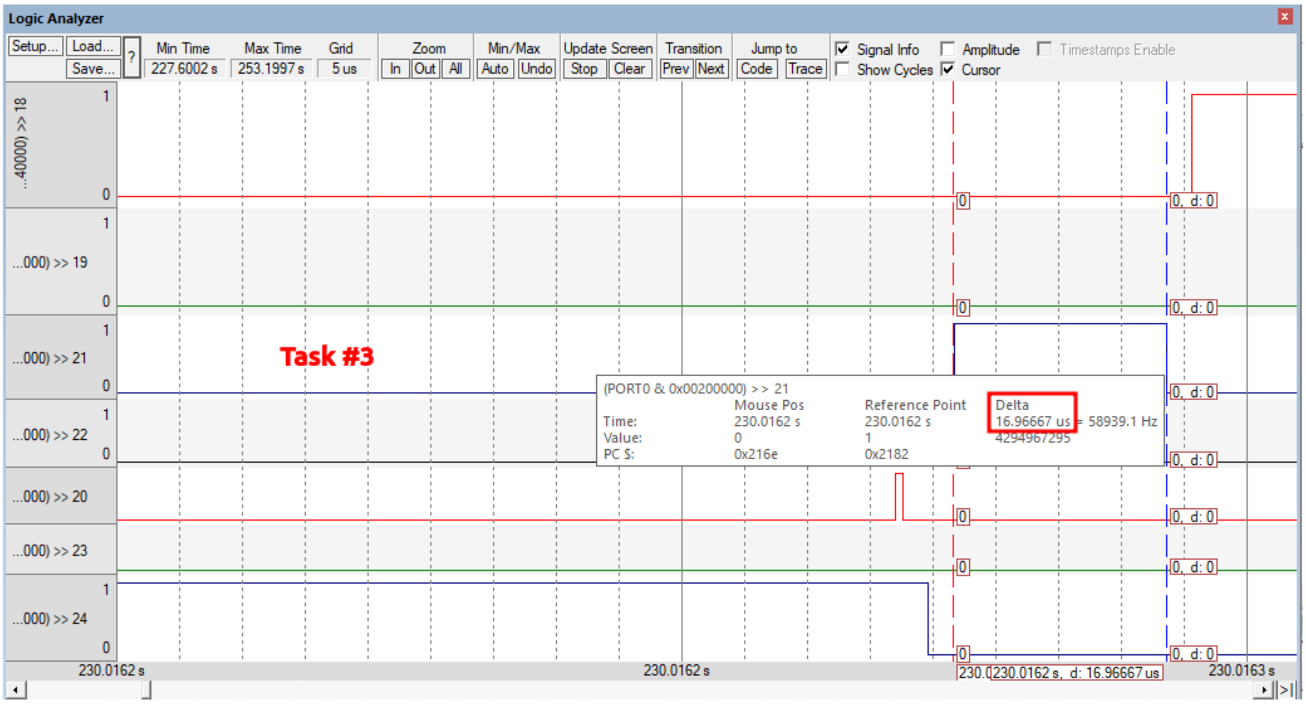
Task 1 (Button 1)



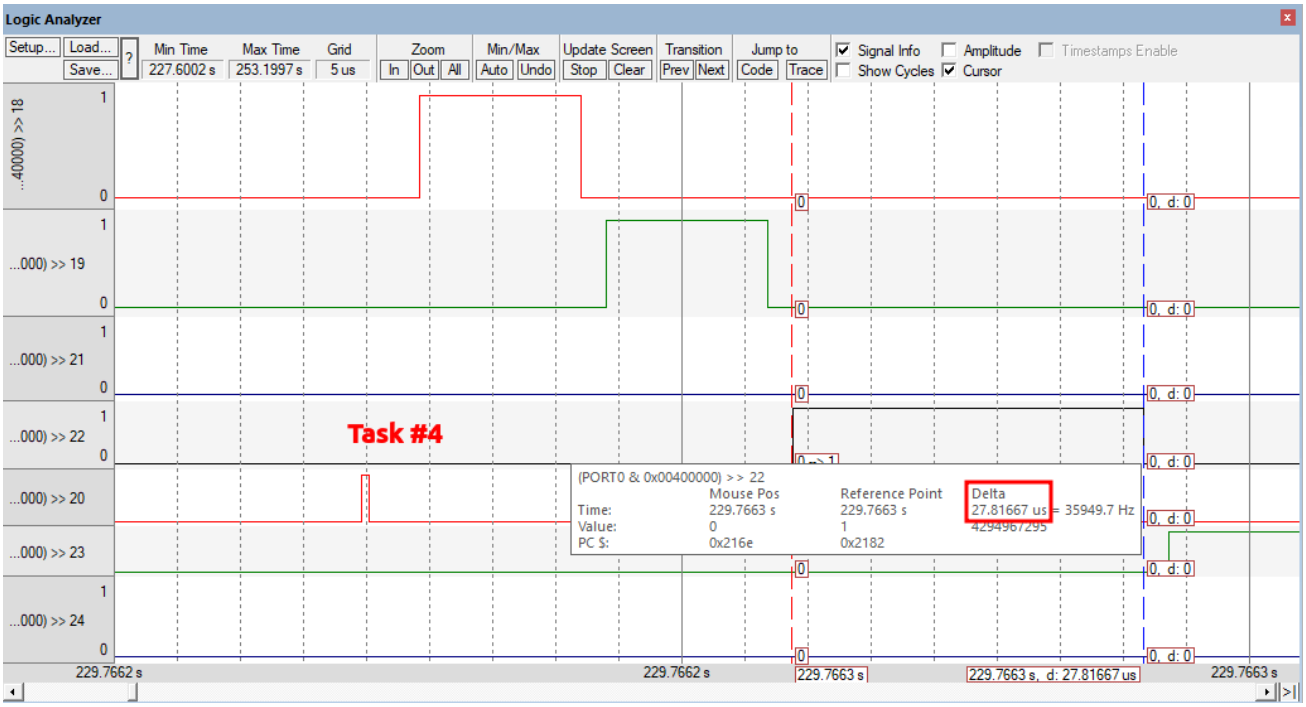
Task 2 (Button 2)



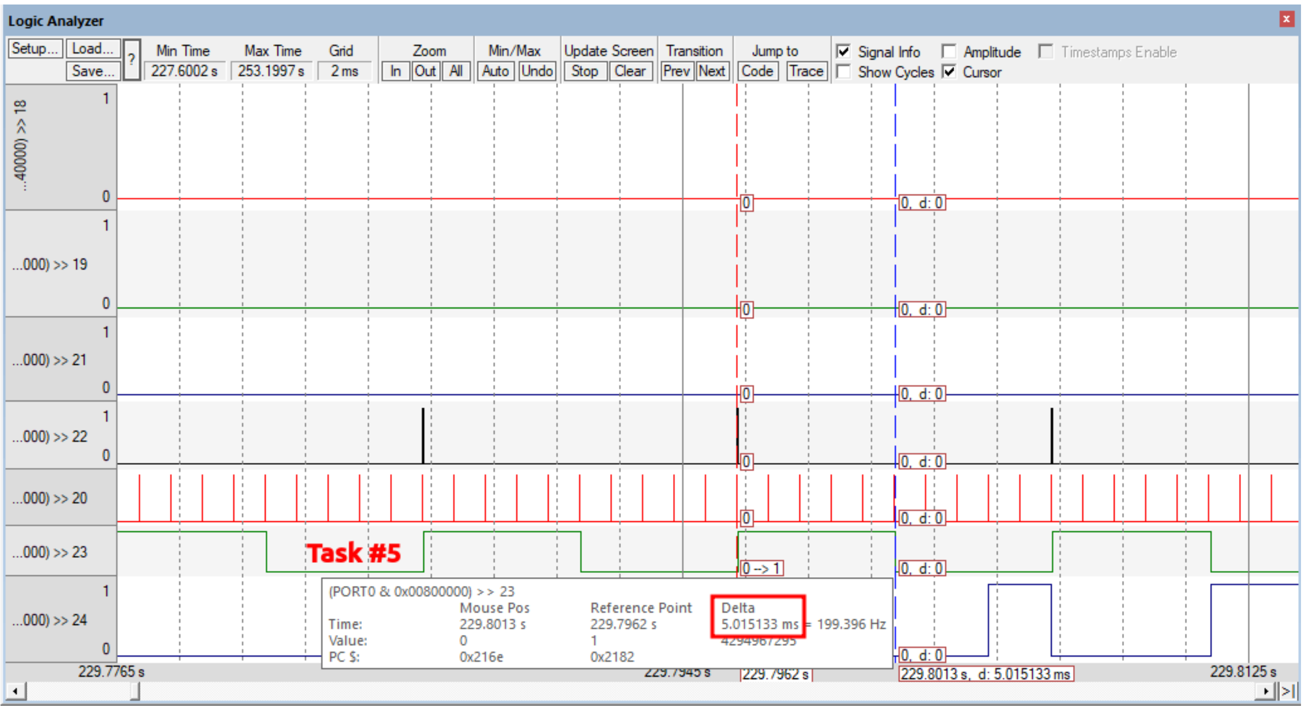
Task 3 (Periodic Transmitter)



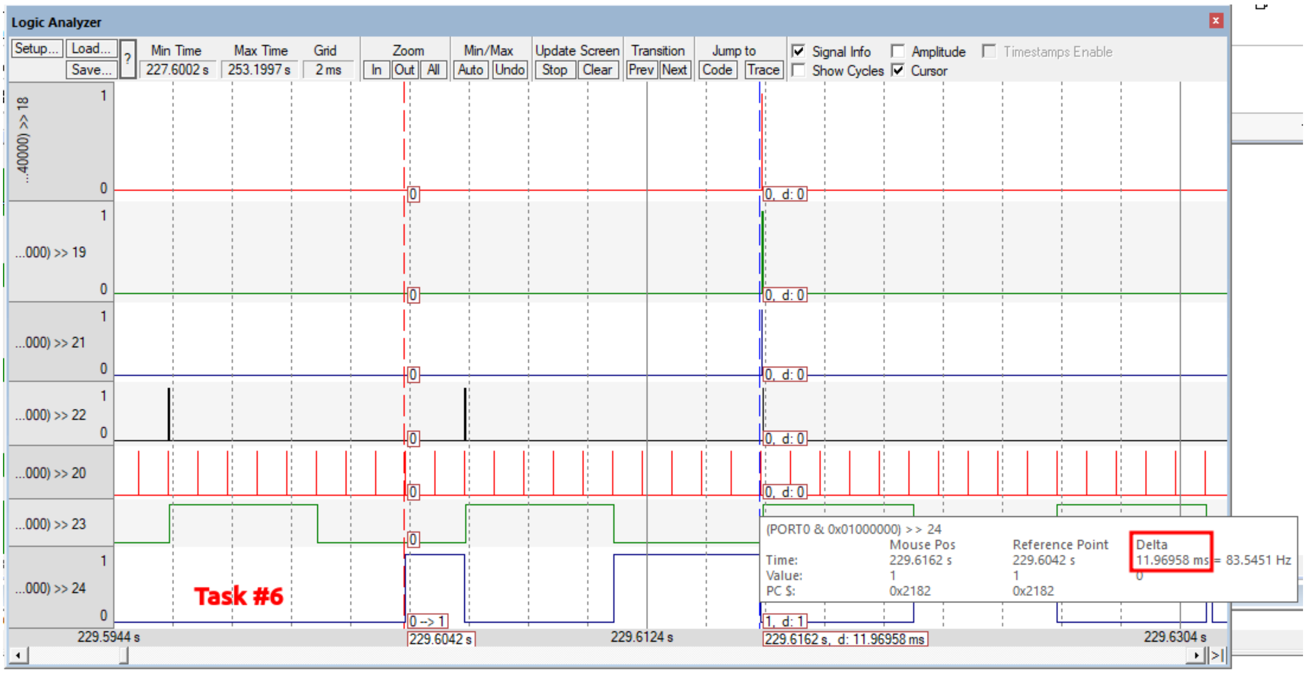
Task 4 (UART Receiver)



Task 5 (Load 1)



Task 6 (Load 2)



Task Name	Periodicity	Priority	Execution Time	Deadline
Button 1	50 ms	1	12.5 µs	50 ms
Button 2	50 ms	1	12.5 µs	50 ms
Periodic TX	100 ms	1	17 µs	100 ms
UART RX	20 ms	1	15 µs	20 ms
Load 1	10 ms	1	5 ms	10 ms
Load 2	100 ms	1	12 ms	100 ms

2.2. Hyper Period

$$Hyperperiod \ = \ LCM(Pi) \ = \ LCM(10,\ 20,\ 50,\ 50,\ 100,\ 100) \ = \ 100\ Ticks$$

2.3. CPU Load

$$CPU\ Load \ = \ \frac{R\ (requirements)}{C\ (capacity)} = \frac{0.015 * 5 + 5 * 10 + 12 + 2 * 0.025 * 2 + 0.017}{100} \ = \ 62.2\%$$

3. URM

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

Where 'C' is the Execution Time, 'P' is the Periodicity and 'n' is the number of tasks.

$$U\ =\ \frac{.015}{20}\ +\ \frac{5}{10}\ +\ \frac{12}{100}\ +\ \frac{.012}{50}\ +\ \frac{.012}{50}\ +\ \frac{.017}{100}\ =\ 0.676$$

$$URM\ =\ 6\ \times\ (2^{\frac{1}{6}}\ -\ 1)\ =\ 0.735$$

$$U\ <\ URM\ \rightarrow\ The\ system\ is\ schedulable\ for\ Rate\ Monotonic\ Schedulers.$$

4. Time Demand Analysis

$$W_i(t)\ =\ E_i + \sum_{k=1}^{i-1}(\frac{t}{P_k})E_k \qquad for\ 0\ <\ t\ \le\ p_i$$

Where:

- W: Worst response time
- E: execution time
- P: periodicity
- t: time instance

Task 1 (Button 1)

$$Time\ Provided\ (T_p)\ =\ Task\ deadline\ =\ 50\ ms$$

$$Time\ Needed\ (T_n)\ =\ W(50)\ =\ .0126\ +\ (\frac{50}{10})\ \times\ 5\ +\ (\frac{50}{20})\ \times\ 0.015\ +\ (\frac{50}{50})\ \times\ 0.0125$$

$$=\ 25.07\ ms$$

$$T_p\ >\ T_n\ \rightarrow\ Button\ 1\ task\ is\ schedulable$$

Task 2 (Button 2)

$$Time\ Provided\ (T_p)\ =\ Task\ deadline\ =\ 50\ ms$$

$$Time\ Needed\ (T_n)\ =\ W(50)\ =\ .0125\ +\ (\frac{50}{10})\ \times\ 5\ +\ (\frac{50}{20})\ \times\ 0.015\ =\ 25ms$$

$$T_p\ >\ T_n\ \rightarrow\ Button\ 2\ task\ is\ schedulable$$

Task 3 (Periodic Transmitter)

$$Time\ Provided\ (T_p)\ =\ Task\ deadline\ =\ 100\ ms$$

$$Time\ Needed\ (T_n) = W(100) = .017 + (\frac{100}{50}) \times .0126 + (\frac{100}{10}) \times 5$$

$$+ (\frac{100}{20}) \times .015 + (\frac{100}{50}) \times .0125 = 50.1\ ms$$

$$T_p > T_n \rightarrow Periodic\ Transmitter\ task\ is\ schedulable$$

Task 4 (UART Receiver)

$$Time\ Provided\ (T_p) = Task\ deadline = 20\ ms$$

$$Time\ Needed\ (T_n) = W(20) = .015 + (\frac{20}{10}) \times 5 = 10.015\ ms$$

$$T_p > T_n \rightarrow UART\ Receiver\ task\ is\ schedulable$$

Task 5 (Load 1)

$$Time\ Provided\ (T_p) = Task\ deadline = 10\ ms$$

$$Time\ Needed\ (T_n) = W(10) = 5 + 0 = 5\ ms$$

$$T_p > T_n \rightarrow Load\ 1\ task\ is\ schedulable$$

Task 6 (Load 2)

$$Time\ Provided\ (T_p) = Task\ deadline = 100\ ms$$

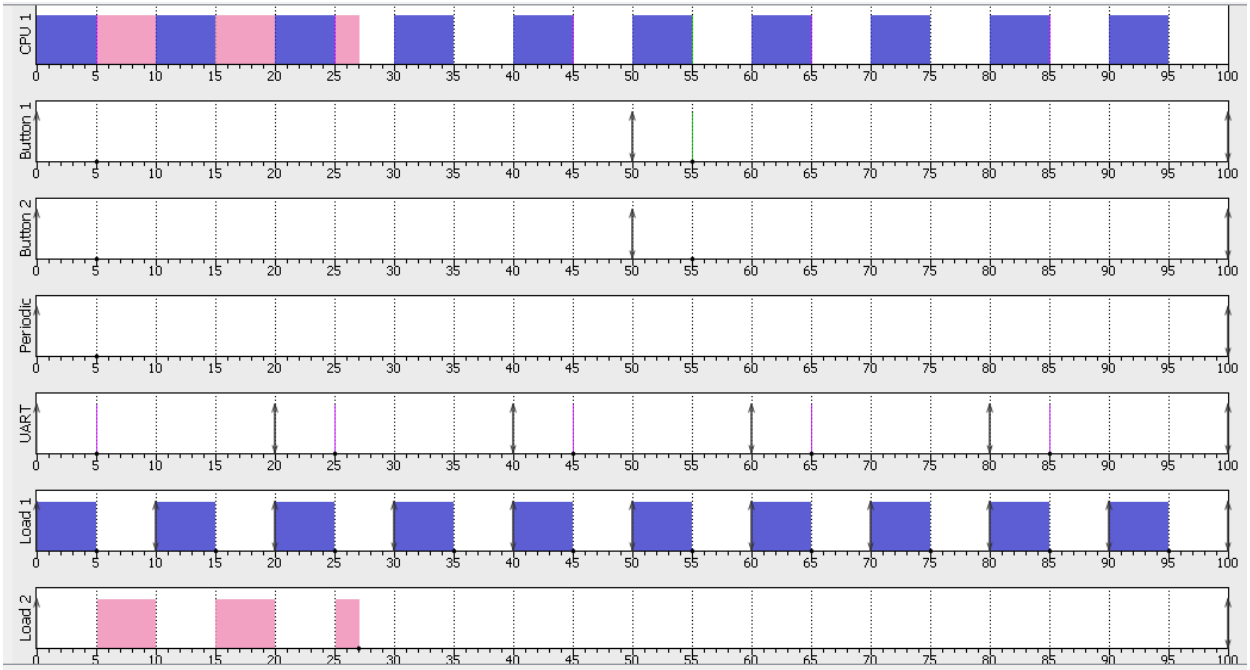
$$Time\ Needed\ (T_n) = W(100) = 12 + (\frac{100}{100}) \times .017 + (\frac{100}{50}) \times .0126 + (\frac{100}{10}) \times 5$$

$$+ (\frac{100}{20}) \times .015 + (\frac{100}{50}) \times .0125 = 62.1\ ms$$

$$T_p > T_n \rightarrow Load\ 2\ is\ schedulable$$

$$All\ tasks\ are\ schedulable \rightarrow System\ is\ schedulable$$

5. SIMSO



	Total load	Payload	System load
CPU 1	0.6214	0.6214	0.0000
Average	0.6214	0.6214	0.0000

Conclusion: Tasks are schedulable.

6. KIEL SIMULATOR

6.1. CPU Load Using Timer 1 and Trace Macros

For calculating CPU load using trace macros, we followed the next steps:

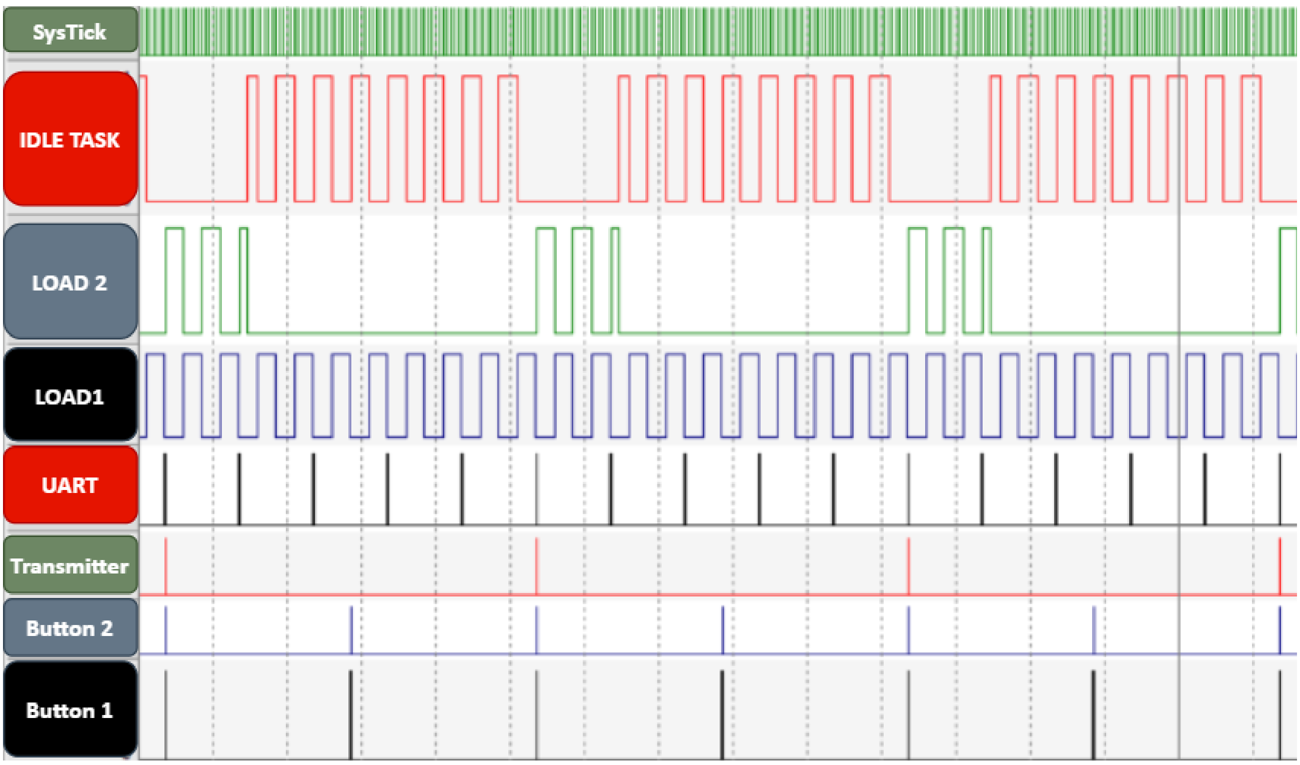
- In the task switch in macro, we calculated time in for the task that called the macro, since it has just started.
- In the task switch-out macro, we calculated time out for the task called the macro, then calculated the total time for that task by subtracting time in from time out for the same task.
- Then initialize the system time with the current time.
- Then for calculating CPU load we divided the summation of all tasks' total time by system time
- The last three steps will be inside the task switch-out macro.

Here is a screenshot for the CPU load for our application using trace macros.

The CPU load is kept changing between 64% - 62%

Conclusion: Actual CPU load is as expected.

6.2. Application Execution Using GPIOs and Trace Macros



Conclusion: Tasks are schedulable.

6.3. uxTaskGetSystemState

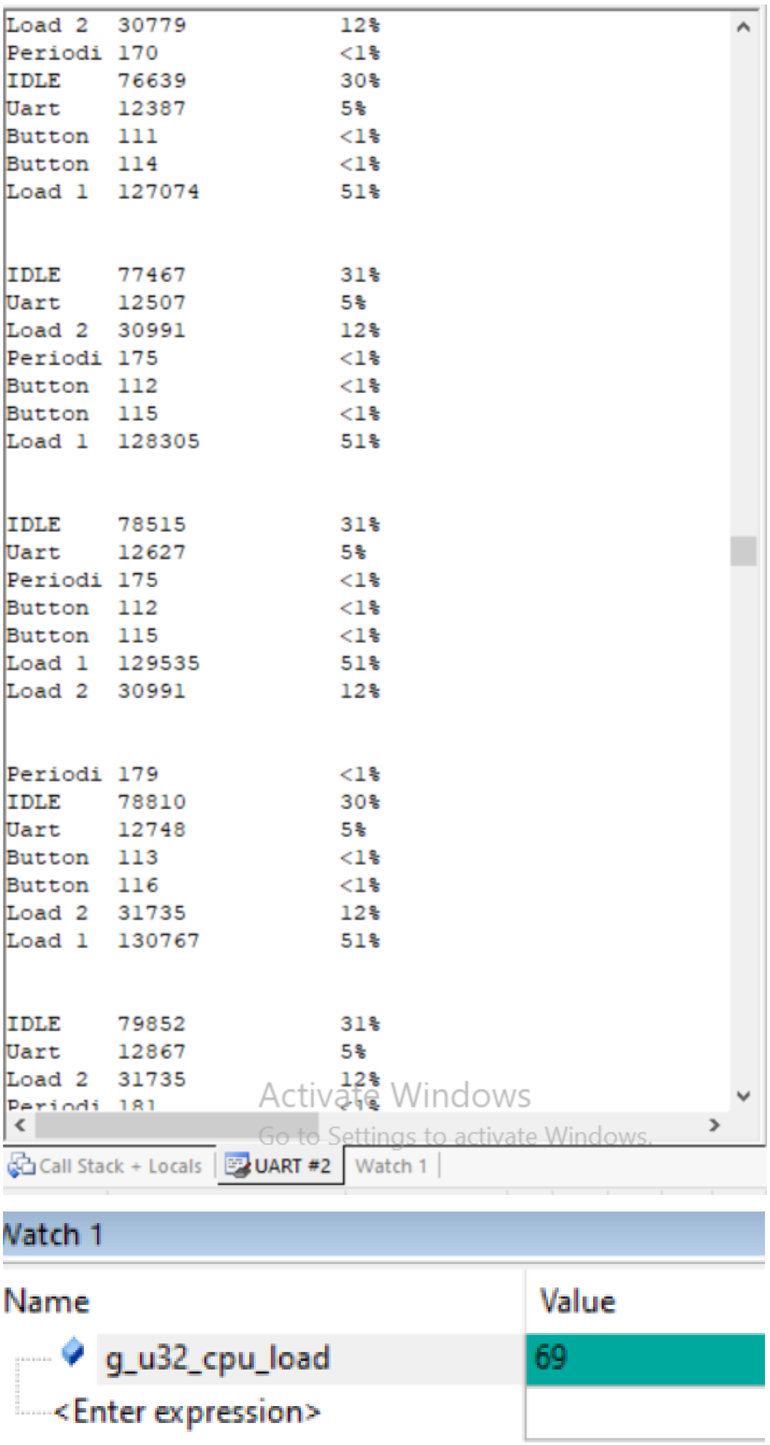
6.3.1. Implementation

As in the next code you find that while using EDF it doesn't declare the queue which will be pointing to the ready list because EDF ready list each element is pointing to an only one task.
Then update uxTasks with the number of tasks ready in the EDF ready list.

```
UBaseType_t uxTaskGetSystemState( TaskStatus_t * const pxTaskStatusArray, const
UBaseType_t uxArraySize, uint32_t * const pulTotalRunTime )
{
  UBaseType_t uxTask = 0;
#if ( configUSE_EDF_SCHEDULER == 0 )
  UBaseType_t uxQueue = configMAX_PRIORITIES;
#endif

  vTaskSuspendAll();
  {
       /* Is there a space in the array for each task in the system? */
       if( uxArraySize >= uxCurrentNumberOfTasks )
       {
       /* Fill in a TaskStatus_t structure with information on each
             task in the Ready state. */
        #if (configUSE_EDF_SCHEDULER == 0)
        do
        {
          uxQueue--;
          uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &(
pxReadyTasksLists[ uxQueue ] ), eReady );
        } while( uxQueue > ( UBaseType_t ) tskIDLE_PRIORITY );
        #else
          uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &(
xReadyTasksListEDF), eReady );
        #endif
```

6.3.2. Verifying



As you can see CPU load while using TaskGetRunTimeStats is 69% since 100% - IDLE utilization = 100% - 31% = 69%, however when I was using the UART to display the states of the buttons the CPU load was 64% that's because when using get runtime stats the UART tends to display much more elements, so the load is bigger than just printing button's states.