Ministry of Communications and Information Technology

Sprints

UDACITY

## Embedded System Advanced Track

## EgFWD-Aug Cohort,2022

## Real Time Operating System Graduation Project

# Implementing EDF Scheduler in FreeRTOS

Mohamed Hassaneen Ahmed

September, 2022

**Content:**

## Introduction:

Embedded systems technological evolution has made the execution of increasingly complex applications possible. Due to this increasing complexity, the adoption of an operating system to manage the interaction between tasks and their scheduling is becoming preferable and even necessary, also for little embedded systems.

FreeRTOS is a real time operating system specially developed for small embedded systems. After an in-depth description of the priority-based scheduler adopted by FreeRTOS, a new scheduler is based on the well-known Earliest Deadline First algorithm (EDF).

EDF 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.

The algorithm assigns priorities to tasks in a simple way: the priority of a task is inversely proportional to its absolute deadline; In other words, the highest priority is the one with the earliest deadline. In case of two or more tasks with the same absolute deadline, the highest priority task among them is chosen random.

The algorithm is suited to work in an environment where these assumptions applies:

1. The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
2. Deadlines consist of run-ability constraints only, i.e. each task must be completed before the next requests for it occurs.
3. The tasks are independent in that requests for a certain task do not depend on the initialization or the completion of requests for other tasks.
4. Run-time for each task is constant for that task and does not vary with time. Run-time refers to the time which is taken by a processor to execute the task without interruption.
5. Any non-periodic tasks in the system are special; they are initialization or failure recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

**A task set of periodic tasks is schedulable by EDF if and only if:**

$$U = \sum_{i=1}^{N} \frac{Ci}{Ti} \leq \quad 1$$

**Ci : Execution Time**

**Ti : Deadline = Periodicity**

## Project Specification:

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| **Read a thesis and implement the required changes** | The following thesis discuss how to implement an EDF scheduler using FreeRTOS.<br><br>1. Download the following thesis: "Implementation and Test of EDF and LLREF Schedulers in FreeRTOS".<br>2. Read chapter 2 : "FreeRTOS Task Scheduling". This is an important chapter to build a profound base before starting the project.<br>3. Read chapter 3 : "EDF Scheduler". This chapter is the main chapter you will use to implement the EDF scheduler using FreeRTOS.<br>4. Watch the final project explanation video to further understand the thesis and the FreeRTOS dependencies.<br>5. Implement the changes mentioned in chapter 3.2.2 : "Implementation in FreeRTOS". The changes will be implemented in tasks.c source file only.<br><br>For this criteria please deliver the following:<br><br>Tasks.c source file with changes implemented from chapter 3.2.2 from the thesis" |
| **Implement the missing changes from the thesis** | Inorder for the EDF scheduler to work correctly, you still need to implement some changes that are not mentioned in the thesis:<br>"1. In the ""prvIdleTask"" function:<br><br>Modify the idle task to keep it always the farest deadline"<br><br>"2. In the ""xTaskIncrementTick"" function:<br><br>In every tick increment, calculate the new task deadline and insert it in the correct position in the EDF ready list"<br><br>"3. In the ""xTaskIncrementTick"" function:<br><br>Make sure that as soon as a new task is available in the EDF ready list, a context switching should take place. Modify preemption way as any task with sooner deadline must preempt task with larger deadline instead of priority" |

| | |
|---|---|
| | "For this criteria please deliver the following:<br><br>Tasks.c source file only with the changes mentioned above implemented" |
| **Implement 4 tasks using EDF scheduler** | Inorder to verify the EDF scheduler, you need to implement an application:<br><br>"1. Create 4 tasks with the following criteria:<br><br>• Task 1: ""Button_1_Monitor"", {Periodicity: 50, Deadline: 50}<br>This task will monitor rising and falling edge on button 1 and send this event to the consumer task. (Note: The rising and failling edges are treated as separate events, hence they have separate strings).<br>• Task 2: ""Button_2_Monitor"", {Periodicity: 50, Deadline: 50}<br>This task will monitor rising and falling edge on button 2 and send this event to the consumer task. (Note: The rising and failling edges are treated as separate events, hence they have separate strings).<br>• Task 3: ""Periodic_Transmitter"", {Periodicity: 100, Deadline: 100}<br>This task will send preiodic string every 100ms to the consumer task.<br>• Task 4: ""Uart_Receiver"", {Periodicity: 20, Deadline: 20}<br>This is the consumer task which will write on UART any received string from other tasks<br>"<br><br>"2. Add a 5th and 6th task to simulate a heavier load:<br><br>• Task 5: ""Load_1_Simulation"", {Periodicity: 10, Deadline: 10}, Execution time: 5ms<br>•<br>Task 6: ""Load_2_Simulation"", {Periodicity: 100, Deadline: 100}, Execution time: 12ms<br><br>These two tasks shall be implemented as en empty loop that loops X times. You shall determine the X times to achieve the required execution time mentioned above. (Hint: In run-time use GPIOs and logic analyzer to determine the execution time)"<br><br>Implement all the tasks mentioned above in the same main.c source file. |

| | |
|---|---|
| | "For this criteria please deliver the following:<br><br>A (maximum 3min) video showing the system working in run-time using Keil simulation. in this video you shall show how the system is working in run-time according to the requirements.<br><br>Deliver main.c, task.c and freertosconfig.h" |
| **Verifying the system implementation** | Now you should verify your system implementation with the EDF scheduler using the following methods:<br>"1. Using analytical methods calculate the following for the given set of tasks:<br><br>• Calculate the system hyperperiod<br>• Calculate the CPU load<br>• Check system schedulability using URM and time demand analysis techniques (Assuming the given set of tasks are scheduled using a fixed priority rate -monotonic scheduler)<br><br>Note: For all the tasks you should calculate the execution time from the actual implemented tasks using GPIOs and the logic analyzer"<br><br>"2. Using Simso offline simulator, simulate the given set of tasks assuming:<br><br>Fixed priority rate monotonic scheduler "<br><br>"3. Using Keil simulator in run-time and the given set of tasks:<br><br>• Calculate the CPU usage time using timer 1 and trace macros<br>• Using trace macros and GPIOs, plot the execution of all tasks, tick, and the idle task on the logic analyzer"<br><br>"For this criteria please deliver the following:<br><br>A PDF report that includes screenshots from the above verification methods and their results. Your report shall also include a comment on the results of these analysis (Ex: Are the results as expected ?, Does the results indicate a successful implementation ?, etc ...).<br><br>Deliver main.c, task.c and freertosconfig.h" |

## Verification of system implementation:

### Hyper Period:

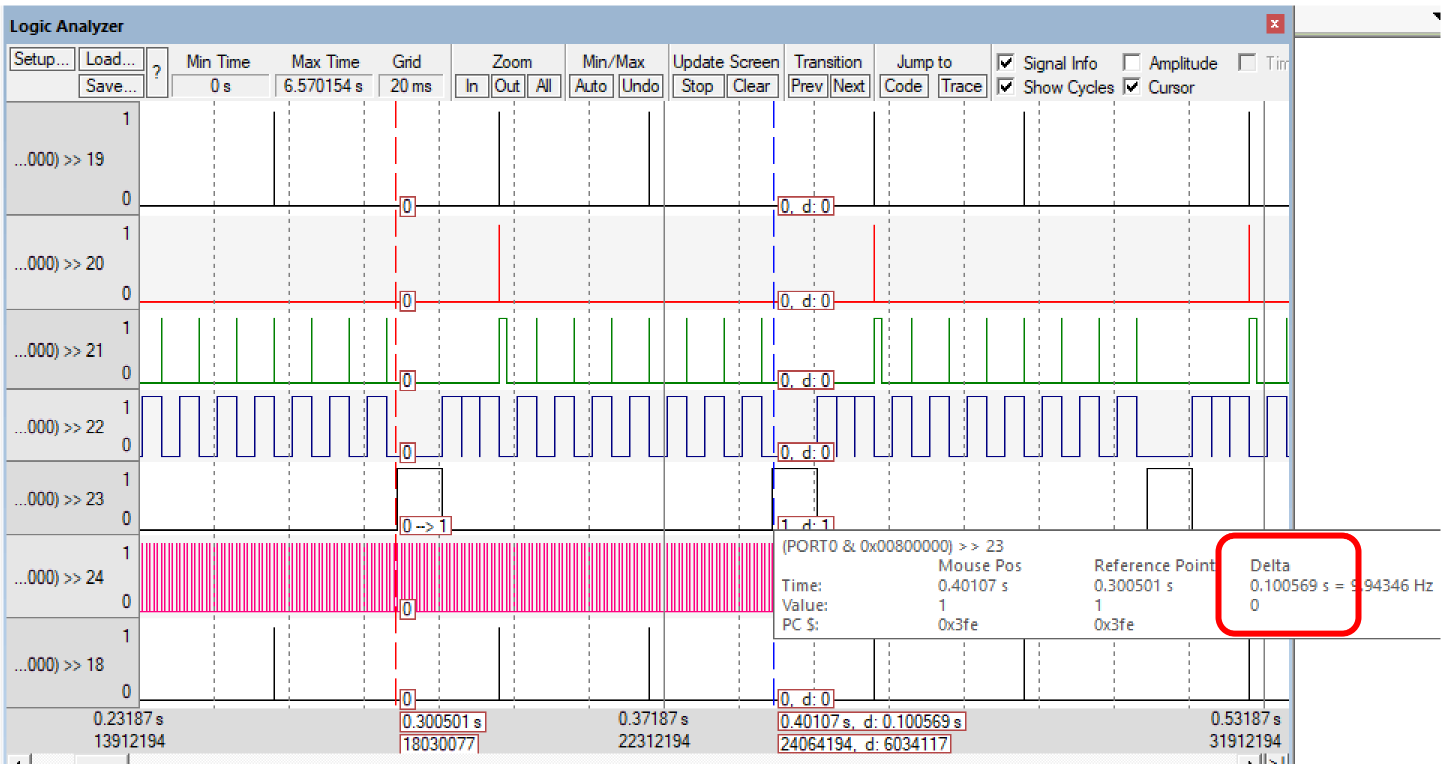


*Figure 1.hyper period*

Hyper-period = 100 ms.

### Execution time:

1- Task_Button__Monitor

Button without elapsing

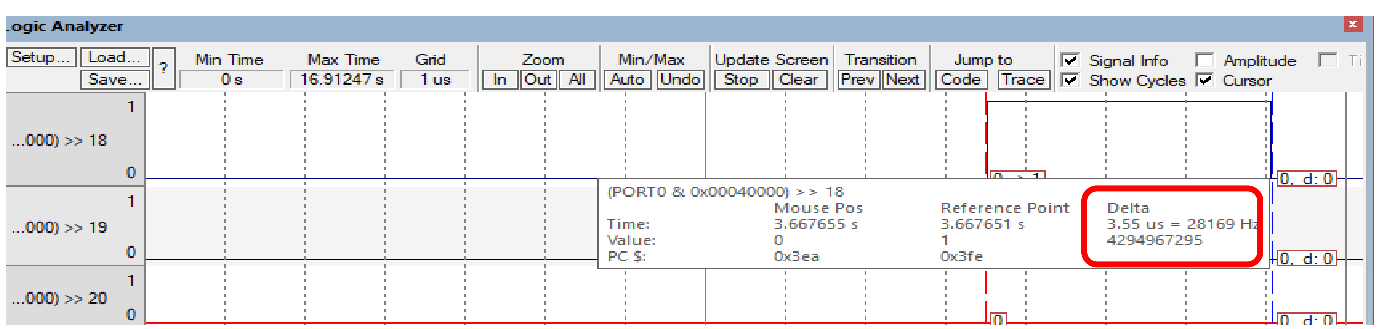


*Figure 2.Button without elapsing*

Button without elapsing = 3.55 microSec

## Button with elapsing

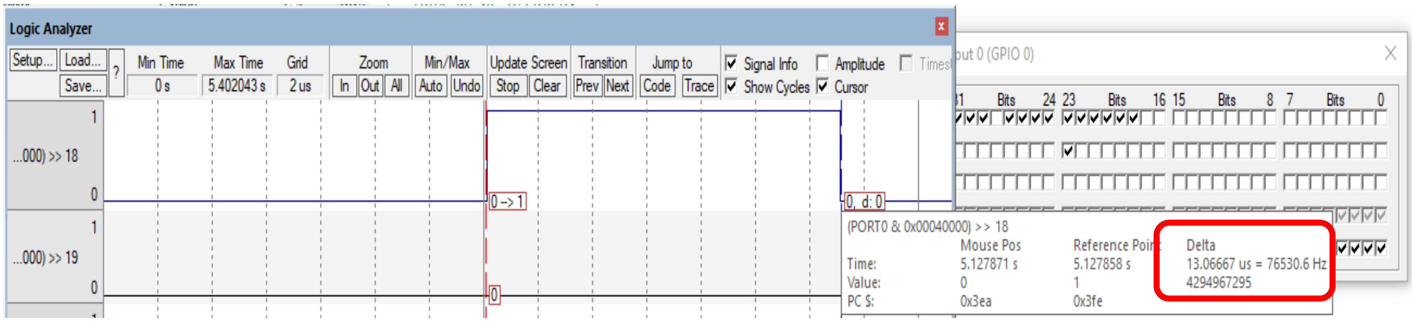


*Figure 3. Button with elapsing*

Button with elapsing =13 microSec

## 2- Task_Periodic_Transmitter

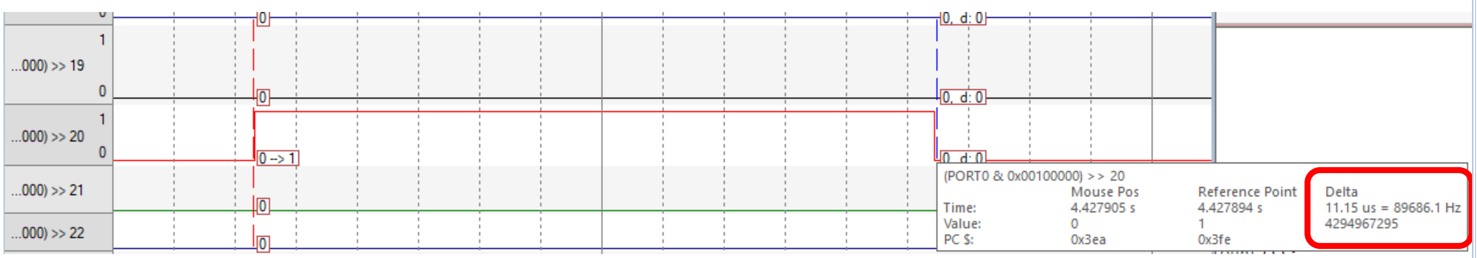


*Figure 4.Task_Periodic_Transmitter*

Task Periodic Transmitter = 11.15 microSec

## 3- Task_Uart_Receiver

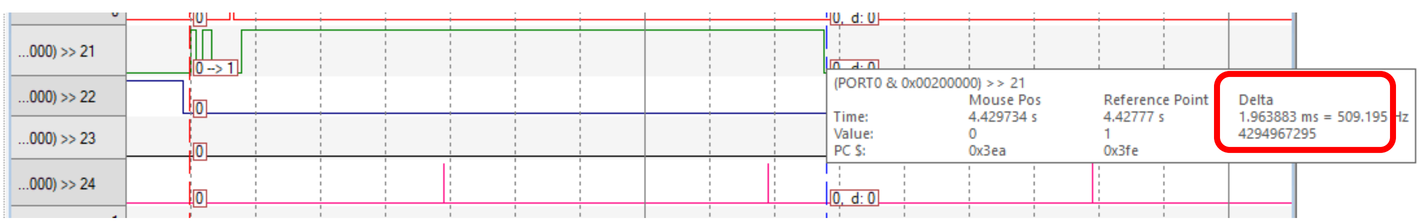


*Figure 5.Task_Uart_Receiver*

Task Uart Receiver = 2 ms

## 4- Load_1_Simulation

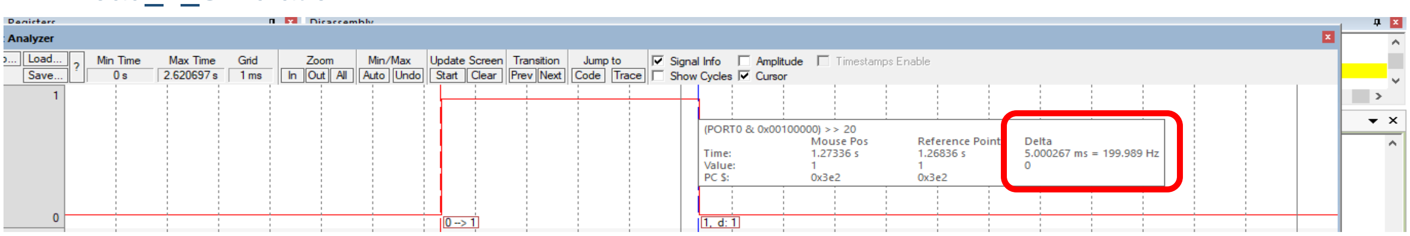


*Figure 6.1-Load_2_Simulation*

Load_1_Simulation = 5 ms.

## 5- Load_2_Simulation

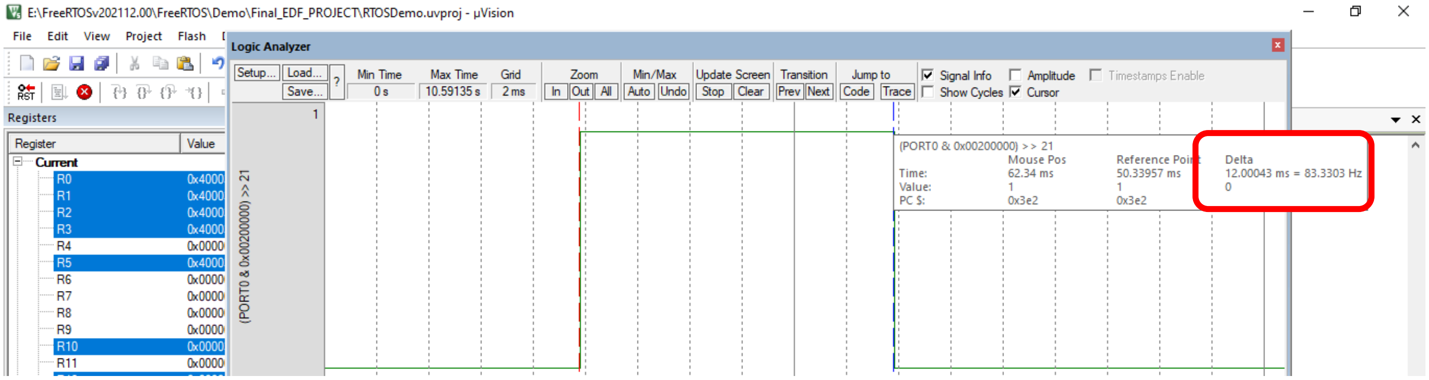


*Figure 7.Load_2_Simulation*

### CPU Load:

| Task | Execution Time (microSec) | Periodicity |
|---|---|---|
| Task_Button_1_Monitor | 13 | 50 |
| Task_Button_2_Monitor | 13 | 50 |
| Task_Periodic_Transmitter | 11.15 | 100 |
| Task Uart Receiver | 2000 | 20 |
| Load_1_Simulation | 5000 | 10 |
| Load_2_Simulation | 120000 | 100 |

Total Execution Time in a Hyper Period = 13*(100/50) + 13 *(100/50) + 11.15*(100/100) +

2000*(100/20)+ 5000*(100/10)+ 12000(100/100)

= 72000 / 1000

= 72 ms.

CPU Load = 72/100 = 72%

CPU Load using Trace and GPIOs:

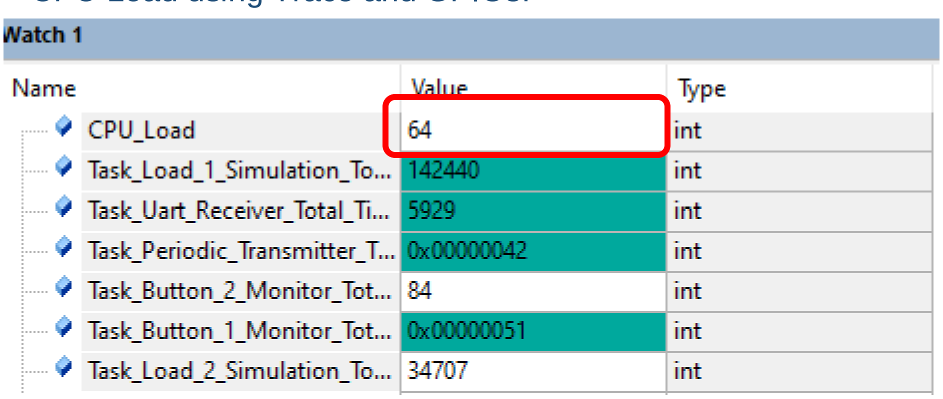


*Figure 8.CPU Load using Trace and GPIOs*

CPU Load = 64%

**Comments:**

Are the results as expected? Why is there a difference between the two results?

Answer:

- Yes, the results are as expected.
- In case of manual calculations, the results are calculated in worst case scenario, for example: the UART task execution time during receiving signals (2ms) is repeated for every 20ms.
- On the other hand, the trace calculations, the results are calculated based on the actual operating system.

  Ex.
  In worst case:
  UART task = 2 * 5 =10 ms.
  In actual:
  UART task = 2 * 1 =2 ms.

  In worst case - In actual = 8 ms.
  The difference represents 8 % of CPU load, if added to the actual calculations (64%) it will give us the same manual calculations (72%).

**Check system schedulability:**

- Using URM

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

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

Assume CPU Load = 64%

URM = 6* (2^ (1/6) - 1) = 73.477 %

Comment: U < URM so that system is guaranteed schedulable

(If we assume CPU load = 72% the system is guaranteed schedulable as well)

- Using time demand analysis techniques
  1. time demand analysis for Load_1_Simulation

     w (10) = 5000 + 0 = 5000 < 10000

     Load_1_Simulation is schedulable.

  2. time demand analysis for Task Uart Receiver

     w(20) = 5000*2 + 2000 = 12000 < 20000

     Task Uart Receiver is schedulable.

  3. time demand analysis for Task_Button_1_Monitor

     w(10) = 5000*5 + 2000*2+13 =29014 < 50000

     Task_Button_1_Monitor is schedulable.

  4. time demand analysis Task_Button_2_Monitor

     w(10) = 5000*5 +2000*2+13*1+13 = 29026 < 50000

     Task_Button_2_Monitor is schedulable.

  5.time demand analysis for Task_Periodic_Transmitter

     w(10) = 5000*10 + 2000*5+13*2+13*2+11.15 = 60063 < 100000

     Task_Periodic_Transmitter is schedulable.

  6.time demand analysis for Load_2_Simulation

     w(10) = 5000*10 + 2000*5+13*2+13*2+11.15 +12000 = 72063 < 100000

     Load_2_Simulation is schedulable.

## Using Simso offline simulator:

- Fixed Priority Scheduler:

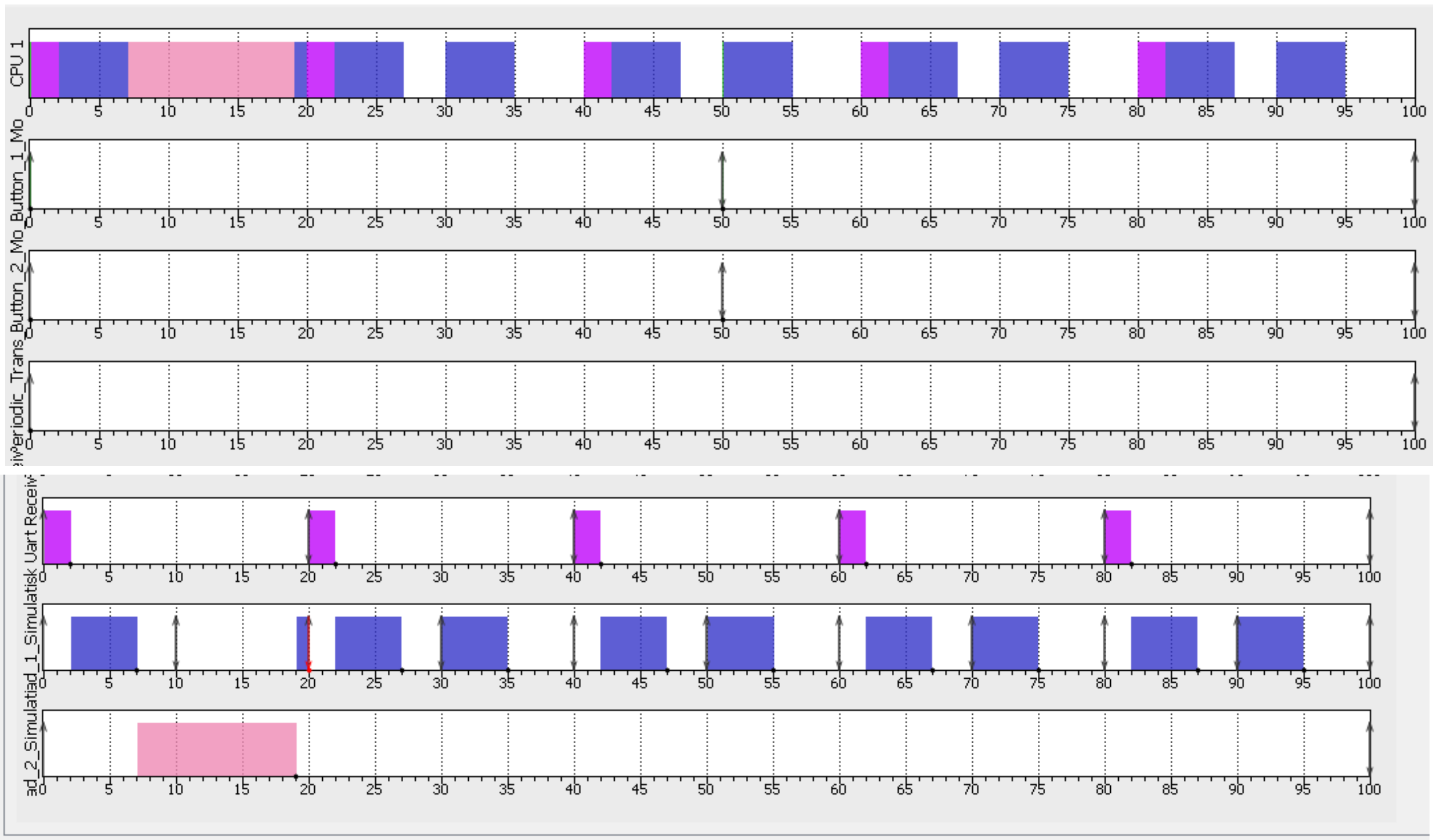


*Figure 9.gantt*

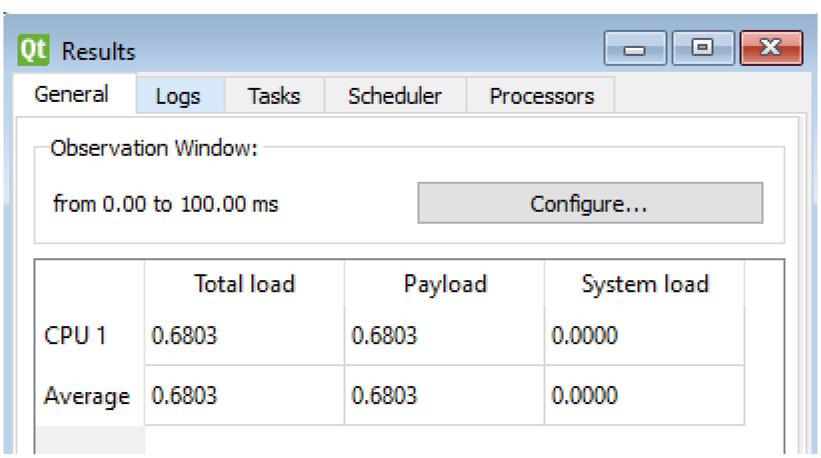


*Figure 10.CPU load*

- EDF and Fixed priority rate monotonic scheduler:

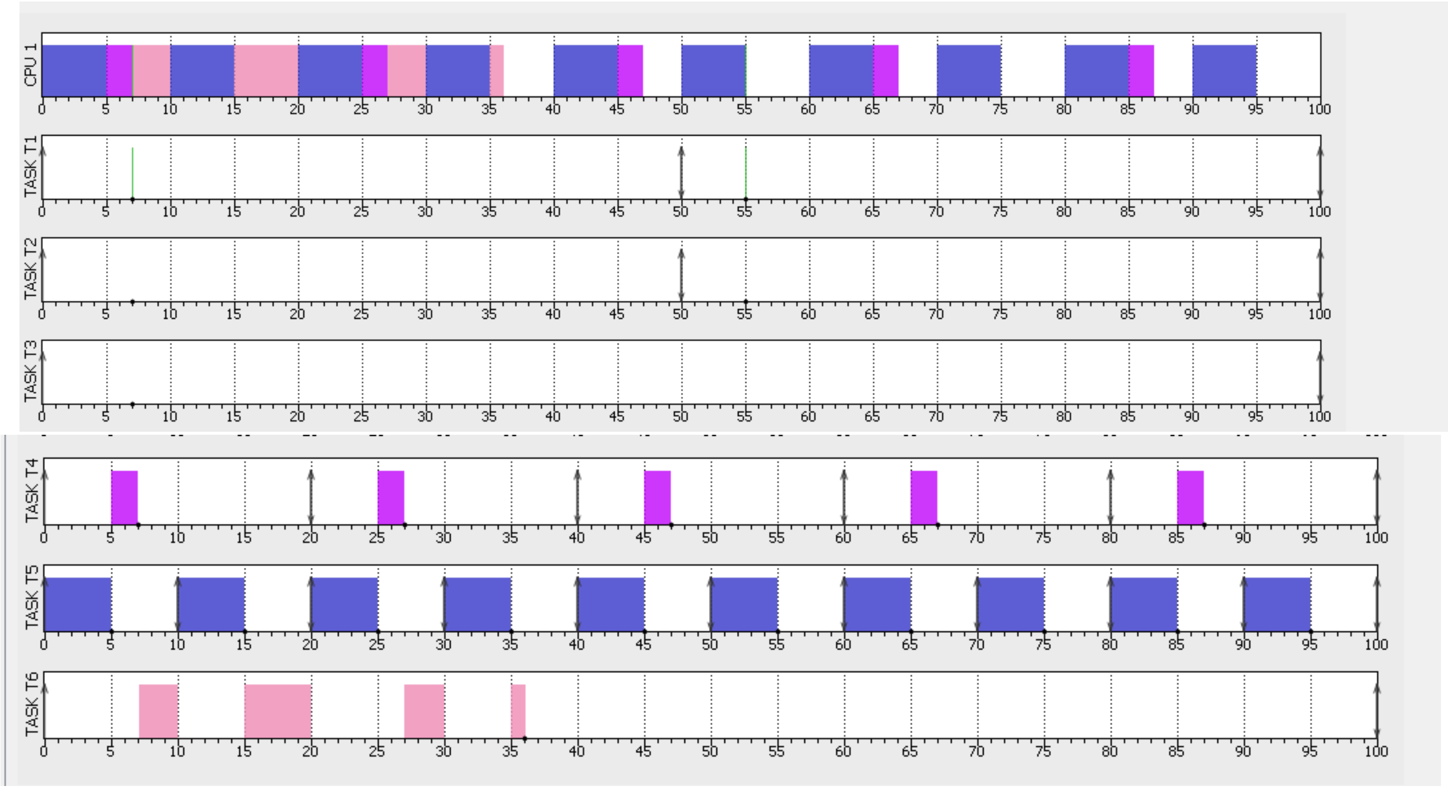


*Figure 11.gantt*

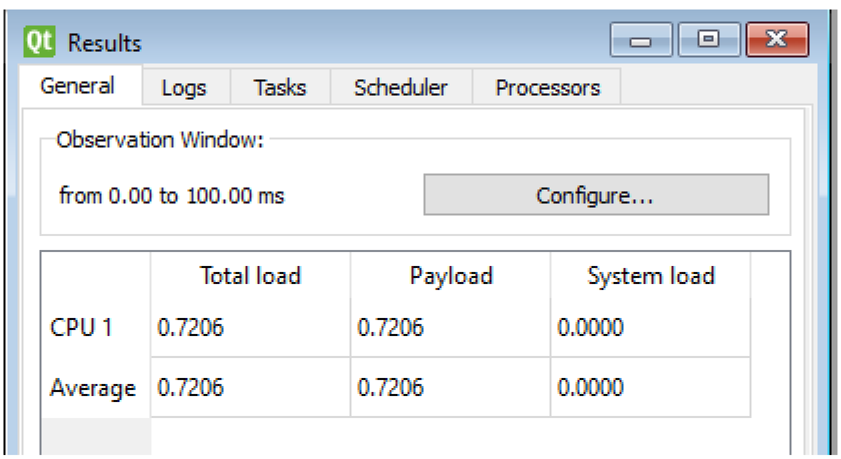

*Figure 12.CPU load*

## Comment:

The CPU load result is the same as manual calculations since Simso calculates the results based on the worst-case scenario.

plotting the execution of all tasks, using trace macros and GPIOs

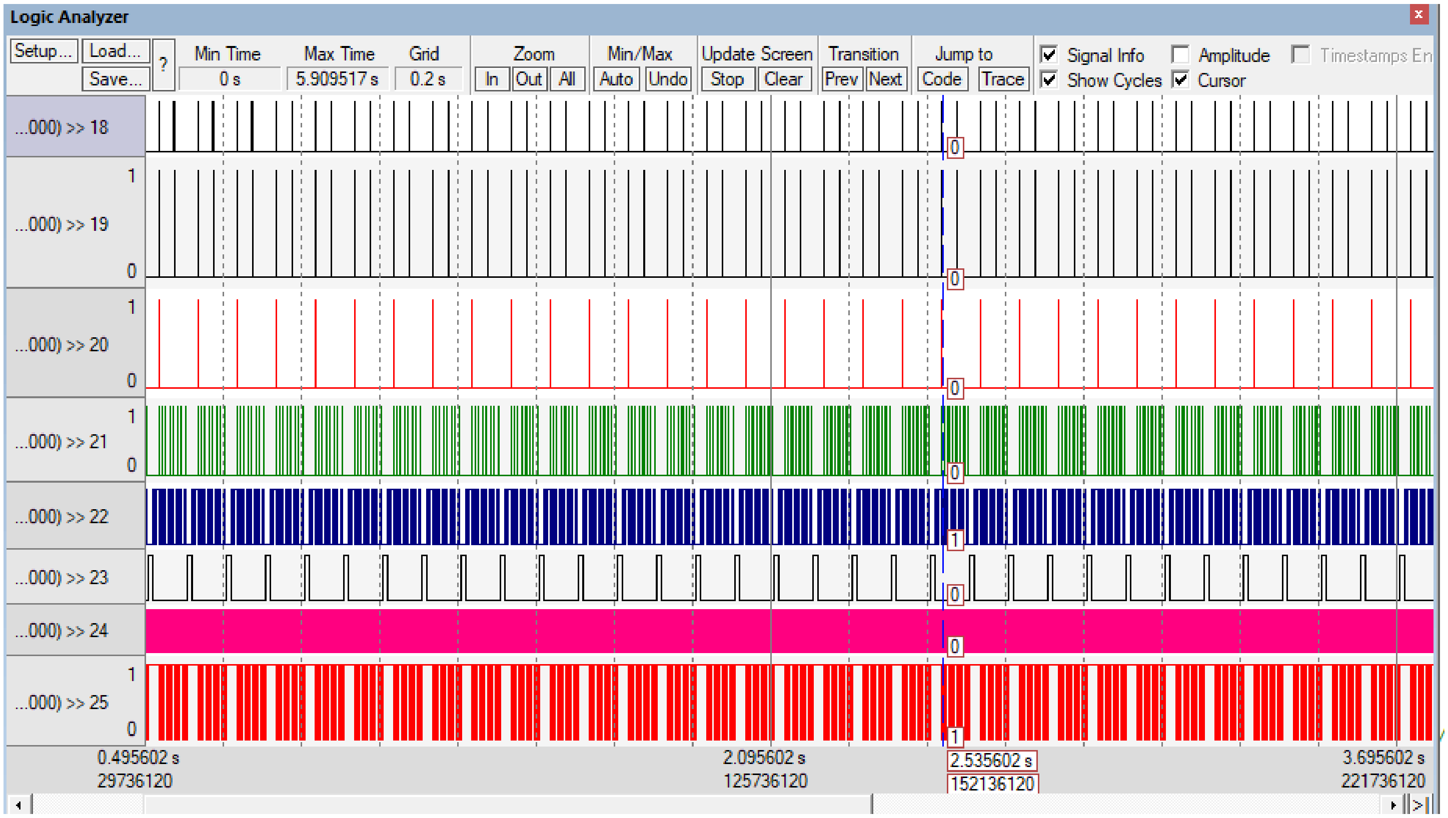


*Figure 13.execution time*

## Implementation:

1- the new Ready List is declared

```
/****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
#if ( configUSE_EDF_SCHEDULER == 1 )
    PRIVILEGED_DATA static List_t xReadyTasksListEDF;        /*< Ready tasks ordered by their deadline. */
#endif

/*-----------------------------------------------------------*/
```

2- prvAddTaskToReadyList() method that adds a task to the Ready List is then modified

```
/****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
#if (configUSE_EDF_SCHEDULER == 0)
#define prvAddTaskToReadyList( pxTCB )                                                          \
    traceMOVED_TASK_TO_READY_STATE( pxTCB );                                                    \
    taskRECORD_READY_PRIORITY( ( pxTCB )->uxPriority );                                         \
    listINSERT_END( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB )->xStateListItem ) ); \
    tracePOST_MOVED_TASK_TO_READY_STATE( pxTCB )
#else
    #define prvAddTaskToReadyList( pxTCB )                                                      \
    traceMOVED_TASK_TO_READY_STATE( pxTCB );                                                    \
        /*( pxTCB )->xStateListItem )-->>  must contain the deadline value */                   \
    vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xStateListItem ) );                       \
    tracePOST_MOVED_TASK_TO_READY_STATE( pxTCB )
#endif
/*-----------------------------------------------------------*/
```

3- A new variable is added in the tskTaskControlBlock structure (TCB)

```
/****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
    #if ( configUSE_EDF_SCHEDULER == 1 )
        TickType_t xTaskPeriod;                /*< Stores the period in tick of the task. > */
    #endif
```

4- xTaskPeriodicCreate() is a modified version of the standard method xTaskCreate()

```
/****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
#if ( configSUPPORT_DYNAMIC_ALLOCATION == 1 )
    #if (configUSE_EDF_SCHEDULER == 0)
    BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                            const char * const pcName, /*lint !e971 Unqualified char types are allowed for strings and single charact
                            const configSTACK_DEPTH_TYPE usStackDepth,
                            void * const pvParameters,
                            UBaseType_t uxPriority,
                            TaskHandle_t * const pxCreatedTask )
```

```
/****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
#if ( configSUPPORT_DYNAMIC_ALLOCATION == 1 )
    #if (configUSE_EDF_SCHEDULER == 1)
    BaseType_t xTaskPeriodicCreate( TaskFunction_t pxTaskCode,
                                    const char * const pcName,          /*lint !e971 Unqualified char types are allowed for strin
                                    const configSTACK_DEPTH_TYPE usStackDepth,
                                    void * const pvParameters,
                                    UBaseType_t uxPriority,
                                    TaskHandle_t * const pxCreatedTask,
                                    TickType_t period                            )
    {
        TCB_t * pxNewTCB;
        BaseType_t xReturn;
```

```
/*******************************************************************
EDF SHCEDULER IMPLEMENTATION
*******************************************************************/
            pxNewTCB->xTaskPeriod = period;

            prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, pxNewTCB, NULL );

            listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), ( pxNewTCB)->xTaskPeriod + xTaskGetTickCount());

            prvAddNewTaskToReadyList( pxNewTCB );

            xReturn = pdPASS;
        }
        else
        {
            xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
        }

        return xReturn;
    }
    #endif  /* configUSE_EDF_SCHEDULER */
#endif /* configSUPPORT_DYNAMIC_ALLOCATION */
/*-----------------------------------------------------------*/
```

5- With the EDF scheduler, the lowest priority behaviour can be simulated by a task having the farest deadline. vTaskStartScheduler() method initializes the IDLE task and inserts it into the Ready List.

```
/*******************************************************************
EDF SHCEDULER IMPLEMENTATION
*******************************************************************/
        #if (configUSE_EDF_SCHEDULER == 1)
            {
            TickType_t initIDLEPeriod = 500;
            xReturn = xTaskPeriodicCreate( prvIdleTask,
                                          configIDLE_TASK_NAME,
                                          configMINIMAL_STACK_SIZE,
                                          (void * ) NULL,
                                           (tskIDLE_PRIORITY | portPRIVILEGE_BIT ),
                                           &xIdleTaskHandle,
                                           initIDLEPeriod );
            }
        #else
            {
        /* The Idle task is being created using dynamically allocated RAM. */
        xReturn = xTaskCreate( prvIdleTask,
                               configIDLE_TASK_NAME,
                               configMINIMAL_STACK_SIZE,
                               ( void * ) NULL,
                               portPRIVILEGE_BIT,  /* In effect ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), but tskIDLE_PRIORITY is zero. */
                               &xIdleTaskHandle ); /*lint !e961 MISRA exception, justified as it is not a redundant explicit cast to all supporte
            }
        #endif /* configUSE_EDF_SCHEDULER */
    }
    #endif /* configSUPPORT_STATIC_ALLOCATION */
```

6- In every tick increment, calculate the new task deadline and insert it in the correct position in the EDF ready list"

```
/*******************************************************************
EDF SHCEDULER IMPLEMENTATION
*******************************************************************/
                    #if ( configUSE_EDF_SCHEDULER == 1 )
                        listSET_LIST_ITEM_VALUE( &( ( pxTCB )->xStateListItem ), ( pxTCB)->xTaskPeriod + xTaskGetTickCount()); /*calculating new d
                    #endif
                    /* Place the unblocked task into the appropriate ready
                     * list. */
                    prvAddTaskToReadyList( pxTCB );
```

7- Make sure that as soon as a new task is available in the EDF ready list, a context switching should take place.

```
/*****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/

                #if ( (configUSE_PREEMPTION == 1 ) && (configUSE_EDF_SCHEDULER == 1 ) )
                    if( (listGET_LIST_ITEM_VALUE( &( ( pxTCB )->xStateListItem )) ) < (listGET_LIST_ITEM_VALUE( &( ( pxCurrentTCB )->xStateListItem ) ) ) )
                        {
                            xSwitchRequired = pdTRUE;
                        }
                #endif
```

8- vTaskSwitchContext() method is in charge to update the pxCurrentTCB pointer to the new running task:

```
/*****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
        #if (configUSE_EDF_SCHEDULER == 0)
         taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-routines too.  Alignment is known t
        #else
         pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );
        #endif
```

9- Modify the idle task to keep it always the farest deadline"

```
/*****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
            #if (configUSE_EDF_SCHEDULER == 0)
                if( listCURRENT_LIST_LENGTH( &( pxReadyTasksLists[ tskIDLE_PRIORITY ] ) ) > ( UBaseType_t ) 1 )
                {
                    taskYIELD();
                }
                else
                {
                    mtCOVERAGE_TEST_MARKER();
                }
        #else

                listSET_LIST_ITEM_VALUE( &( ( pxCurrentTCB)->xStateListItem ), (pxCurrentTCB)->xTaskPeriod + xTaskGetTickCount());
                if( listCURRENT_LIST_LENGTH( &xReadyTasksListEDF ) > ( UBaseType_t ) 1 )
                {
                    taskYIELD();
                }
                else
                {
                    mtCOVERAGE_TEST_MARKER();
                }
        #endif
            }
        #endif /* ( ( configUSE_PREEMPTION == 1 ) && ( configIDLE_SHOULD_YIELD == 1 ) ) */
```

10-the prvInitialiseT askLists() method, that initialize all the task lists at the creation of the first task, is modified adding the initialization of xReadyTasksListEDF:

```
/*****************************************************************
EDF SHCEDULER IMPLEMENTATION
*****************************************************************/
    #if ( configUSE_EDF_SCHEDULER == 1 )
        {
            vListInitialise( &xReadyTasksListEDF );
        }
    #endif
```

## References:

Enrico Carraro, Implementation and Test of EDF and LLREF Schedulers in FreeRTOS, Computing of the University of London and the Diploma of Imperial College, April 2016.