Report on:

EDF Scheduler Implementation in FreeRTOS

Prepared by:

Abanoub Salah Mitry

Submitted To: **Submitted Date:**

Udacity Team

Contents

Introduction

Real-time operating systems (RTOS) are defined as a system that outputs a correct value at the correct time. It can be categorized as follow, *soft real-time* system which can deviate within a specific limits without fatal consequences. The second category is hard real-time system that needs not to deviate from its specified value and time otherwise it result in fatal consequences.

Examples for soft real-time systems: gaming console, digital cameras and mp3 player.
Examples for hard real-time systems: weapons defense system, flight control and any safety critical system.

Implementing RTOS using FreeRTOS as a kernel needs a scheduler algorithm or as it is called policy. FreeRTOS includes round-robin or rate monotonic algorithms preemptive or non-preemptive.

In this project we will implement the Earliest Deadline First algorithm (EDF) with the help of “Implementation and Test of EDF and LLREF Schedulers in FreeRTOS” thesis paper as a reference.

EDF Implementation

By only modifying *tasks.c* file we can add EDF algorithm functionality in freeRTOS.

To conform to the freeRTOS style guideline we will add a configuration variable *configUSE_EDF_SCHEDULER* to include or exclude EDF algorithm code.

```
#define configUSE_TRACE_FACILITY    1
#define configUSE_16_BIT_TICKS      0
#define configIDLE_SHOULD_YIELD     1

#define configUSE_EDF_SCHEDULER     1

#define configQUEUE_REGISTRY_SIZE   0

#define configUSE_TIMERS            0

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES       0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet        0
#define INCLUDE_uxTaskPriorityGet       0
#define INCLUDE_vTaskDelete             0
#define INCLUDE_vTaskCleanUpResources   0
#define INCLUDE_vTaskSuspend            0
#define INCLUDE_vTaskDelayUntil         1
#define INCLUDE_vTaskDelay              0

#if configUSE_TRACE_FACILITY == 1
#define STATS_BUFFER_LENGTH                     250
#define configGENERATE_RUN_TIME_STATS           1
#define configUSE_STATS_FORMATTING_FUNCTIONS    1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()
#define portGET_RUN_TIME_COUNTER_VALUE()        T1TC
#endif
```

<u>First change</u>, we will add a new EDF ready list that includes ready tasks sorted according to the earliest deadline first.

```c

/* E.C. : the new ReadyList */
#if ( configUSE_EDF_SCHEDULER == 1 )
    PRIVILEGED_DATA static List_t xReadyTasksListEDF;   /*< Ready tasks ordered by their deadline. >*/
#endif

```

Then, the *prvInitialiseTaskLists()* method which initialize the task lists before adding any task. So we add the EDF ready task to be initialized also.

```c
static void prvInitialiseTaskLists( void )
{
    UBaseType_t uxPriority;

    for( uxPriority = ( UBaseType_t ) 0U; uxPriority < ( UBaseType_t ) configMAX_PRIORITIES; uxPriority++ )
    {
        vListInitialise( &( pxReadyTasksLists[ uxPriority ] ) );
    }

    vListInitialise( &xDelayedTaskList1 );
    vListInitialise( &xDelayedTaskList2 );
    vListInitialise( &xPendingReadyList );

    /* E.C. */
    #if ( configUSE_EDF_SCHEDULER == 1 )
    {
        vListInitialise( &xReadyTasksListEDF );
    }
    #endif
```

Then, modifying *prvAddNewTaskToReadyList()* to set the earliest deadline task when EDF where used instead of the highest priority task if scheduler is not already running.

```c
            if( xSchedulerRunning == pdFALSE )
            {
                /* E.C. : select current if scheduler is not running */
                #if ( configUSE_EDF_SCHEDULER == 1 )
                {
                    if( pxCurrentTCB->xStateListItem.xItemValue > pxNewTCB->xStateListItem.xItemValue )
                    {
                        pxCurrentTCB = pxNewTCB;
                    }
                    else
                    {
                        mtCOVERAGE_TEST_MARKER();
                    }
                }
                #else
                if( pxCurrentTCB->uxPriority <= pxNewTCB->uxPriority )
                {
                    pxCurrentTCB = pxNewTCB;
                }
                else
                {
                    mtCOVERAGE_TEST_MARKER();
                }
                #endif
            }
```

Also the *prvAddTaskToReadyList()* macro altered to add the deadline value to the *xGenericListIteam*. Since *vListInsert* assumes that *xGenericListItem* contains the next task deadline.

```c

/* E.C. : place the task to the edf ready list */
#if configUSE_EDF_SCHEDULER == 1
    #define prvAddTaskToReadyList( pxTCB )                                                                \
        traceMOVED_TASK_TO_READY_STATE( pxTCB );                                                          \
        vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xStateListItem ) );                             \
        tracePOST_MOVED_TASK_TO_READY_STATE( pxTCB )
#else
    #define prvAddTaskToReadyList( pxTCB )                                                                \
        traceMOVED_TASK_TO_READY_STATE( pxTCB );                                                          \
        taskRECORD_READY_PRIORITY( ( pxTCB )->uxPriority );                                               \
        vListInsertEnd( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB)->xStateListItem ) ); \
        tracePOST_MOVED_TASK_TO_READY_STATE( pxTCB )
#endif
/*-----------------------------------------------------------*/

```

<u>Second change</u>, we will add the period to the TCB as reference. Since to calculate the deadline we need the knowledge of the period.

$$Task_{deadline} = current_{tick} + Task_{period}$$

```c

    /* E.C. : the period of a task */
    #if ( configUSE_EDF_SCHEDULER == 1 )
        TickType_t xTaskPeriod;     /*< Stores the period in tick of the task. > */
    #endif
} tskTCB;

```

Accordingly we need a new method *xTaskPeriodicCreate* to initialize new tasks based on *xTaskGenericCreate* method.

xTaskPeriodicCreate will set the task period that is needed for deadline calculation and the task *xGenericListItem* value that is needed for lists sorting according to earliest deadline.

```c
/*E.C. : task creation function */
#if ( configUSE_EDF_SCHEDULER == 1 )
BaseType_t xTaskPeriodicCreate( TaskFunction_t pxTaskCode,
                                const char * const pcName, /*lint !e971 Unqualified char types are allowed for strings and single
                                characters only. */
                                const configSTACK_DEPTH_TYPE usStackDepth,
                                void * const pvParameters,
                                UBaseType_t uxPriority,
                                TaskHandle_t * const pxCreatedTask,
                                TickType_t period )
{
        ...

        /*E.C. : set the task period */
        pxNewTCB->xTaskPeriod = period;

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

        /*E.C. : insert the deadline to the generic item list before adding the task to the ready list: */
        listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), ( pxNewTCB)->xTaskPeriod + xTaskGetTickCount());

        prvAddNewTaskToReadyList( pxNewTCB );
        xReturn = pdPASS;
    }
    else
    {
        xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
    }

    return xReturn;
}
```

<u>Third change</u>, IDLE task management. Since freeRTOS requires the existence of a ready task at all instances, IDLE task modification is fundamental to the working of our scheduler.

IDLE task is a simple task initialization in *vTaskStartScheduler()* method at the lowest priority, with the EDF scheduler we will create the IDLE task with the farthest deadline possible.

```c

        /*E.C. : the IDLE task Modification */
        #if (configUSE_EDF_SCHEDULER == 1)
        {
        xReturn = xTaskPeriodicCreate( 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 supported compilers. */
                                       portMAX_DELAY );
        }
        #else
        {
        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 supported compilers. */
        }
        #endif
    }
    #endif /* configSUPPORT_STATIC_ALLOCATION */

```

Then, setting in xTaskIncrementTick() method the IDLE task deadline to the maximum value to keep the IDLE task the farthest in the EDF ready list.

```c
    /* E.C. : incremet the idle task deadline */
    #if ( configUSE_EDF_SCHEDULER == 1 )
    {
        if( xIdleTaskHandle->xStateListItem.xItemValue < portMAX_DELAY )
        {
            xIdleTaskHandle->xStateListItem.xItemValue = portMAX_DELAY;
        }
    }
    #endif

```

<u>Fourth change</u>, In the xTaskIncrementTick() method we update the current ready task deadline.

```
/* E.C. : update dealine of the task */
#if ( configUSE_EDF_SCHEDULER == 1 )
{
    listSET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ), listGET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ) ) + pxTCB->xTaskPeriod );
}
#endif
```

In the same method we check if the current task deadline is farthest than the unblocked task and decide if a task switch is required.

```
/* E.C. : the period of a task */
#if ( configUSE_EDF_SCHEDULER == 1 )
{
if( listGET_LIST_ITEM_VALUE( &( pxCurrentTCB->xStateListItem ) ) > listGET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ) ) )
{
    xSwitchRequired = pdTRUE;
}
else
{
    mtCOVERAGE_TEST_MARKER();
}
}
#else
{
if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
{
    xSwitchRequired = pdTRUE;
}
else
{
    mtCOVERAGE_TEST_MARKER();
}
}
#endif
```

In the vTaskSwitchContext() method which responsible for switch context mechanism when a running task get suspended or a suspended task with a higher priority than the running task awakes. *vTaskSwitchContext()* updates the **pxCurrentTCB* with the highest priority using *taskSELECT_HIGHEST_PRIORITY_TASK()* macro which is replaced here to update it with the task with the earliest deadline.

```
/* Select a new task to run using either the generic C or port
 * optimised asm code. */
/* E.C. : select a new task to run depending on scheduler */
#if ( configUSE_EDF_SCHEDULER == 1 )
{
pxCurrentTCB = listGET_OWNER_OF_HEAD_ENTRY( &( xReadyTasksListEDF ) );
traceTASK_SWITCHED_IN();
}
#else
{
taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-routines too.  Alignment is known to be fine as the type of the pointer stored and retrieved is the same. */
traceTASK_SWITCHED_IN();
}
#endif
```

Last change, In the uxTaskGetSystemState () method which gets the system states to display this information to the UART. We fill the TaskStatus_t structure depending on the scheduler used.

```c
/* Fill in an TaskStatus_t structure with information on each
 * task in the Ready state. */
/* E.C. : alter uxQueue & pxReadyTasksLists to work for EDF */
#if configUSE_EDF_SCHEDULER == 1
/* To avoid compiler unused variable warning */
(void)uxQueue;
uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &( xReadyTasksListEDF ), eReady );
#else
do
{
    uxQueue--;
    uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ), &( pxReadyTasksLists[ uxQueue ]
    ), eReady );
} while( uxQueue > ( UBaseType_t ) tskIDLE_PRIORITY ); /*lint !e961 MISRA exception as the casts are only
redundant for some ports. */

#endif
```

Example System Used

System specs where extracted from the project specification page on udacity platform and keil simulation traces as follow.

Task	**Period (ms)**	**Deadline (ms)**	**WCET (ms)**	**Priority**
T1	50	50	0.018	3
T2	50	50	0.018	3
T3	100	100	0.017	4
T4	20	20	0.073	2
T5	10	10	5	1
T6	100	100	12	4

The system will be scheduled using EDF algorithm that is implemented by modifying freeRTOS code.

EDF Scheduler feasibility

Analytical Method

$$Hyper\ Period(H) = LCM(T_i) = LCM(50, 50, 100, 20, 10, 100)$$
$$= 100\ ms$$

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} = \frac{0.018}{50} + \frac{0.018}{50} + \frac{0.017}{100} + \frac{0.073}{20} + \frac{5}{10} + \frac{12}{100}$$
$$= 0.62454 < 1$$

$$Urm = n\left(2^{\frac{1}{n}} - 1\right) = 6\left(2^{\frac{1}{6}} - 1\right) = 0.73$$

<u>Time demand analysis</u>:

$$W_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k \text{ for } 0 < t \le P_i$$

According to the task priority and maximum time instant at the end of the period.

For T5: $W_5(10) = 5 + 0 = 5 \le D = 10$

∴T5 is schedulable

For T4: $W_4(20) = 0.073 + 10 = 10.073 \le D = 20$

∴T4 is schedulable

For T1: $W_1(50) = 0.018 + 25 + 0.219 = 25.237 \le D = 50$

∴T1 is schedulable

For T2: $W_2(50) = 0.018 + 25 + 0.219 + 0.018 = 25.255 \le D = 50$

∴T2 is schedulable

For T3: $W_3(100) = 0.017 + 50 + 0.365 + 0.036 + 0.036 = 50.454 \leq D = 100$

$\therefore$T3 is schedulable

For T6: $W_6(100) = 12 + 50 + 0.365 + 0.036 + 0.036 + 0.017 = 62.454 \leq D = 100$

$\therefore$T6 is schedulable

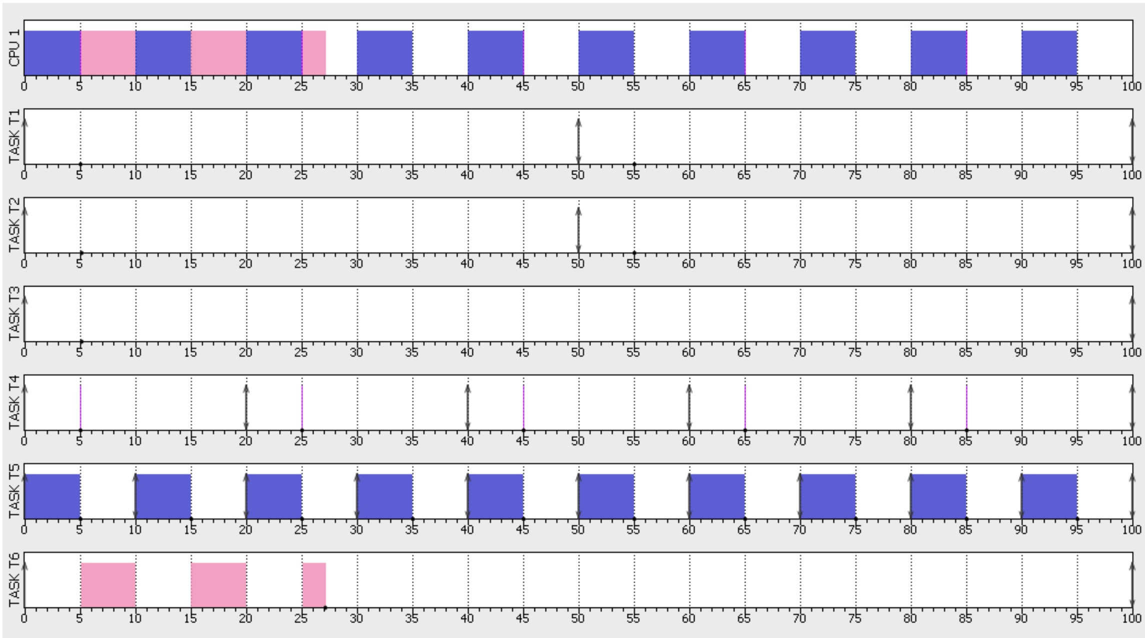
Results:

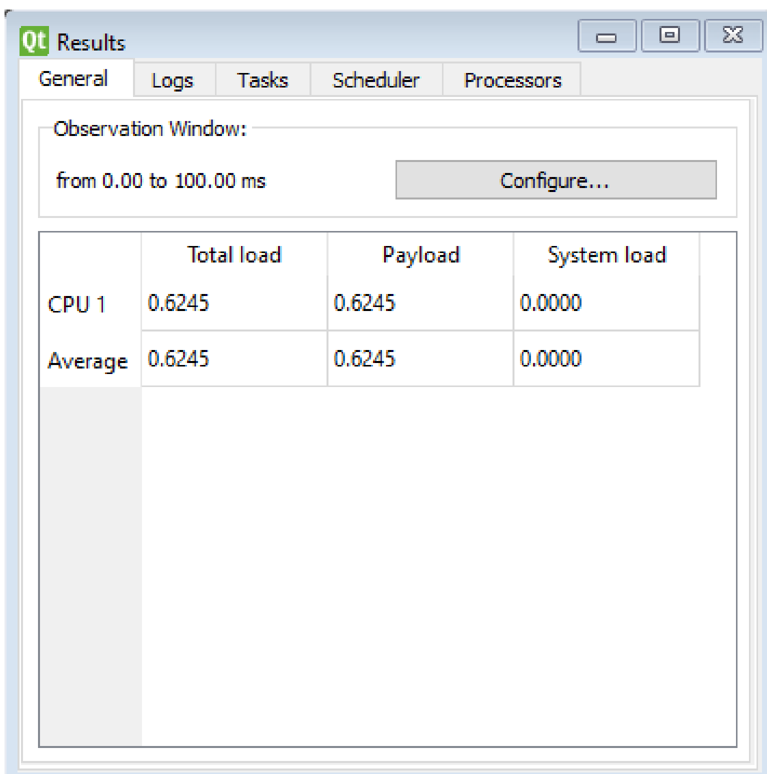
From time demand analysis all the tasks are schedulable.

And $\because U \leq Urm \therefore$ System maybe schedulable using RM algorithm

Also $\because U \leq 1$ $\therefore$ System is feasible to be scheduled with EDF algorithm.

Simso Simulation

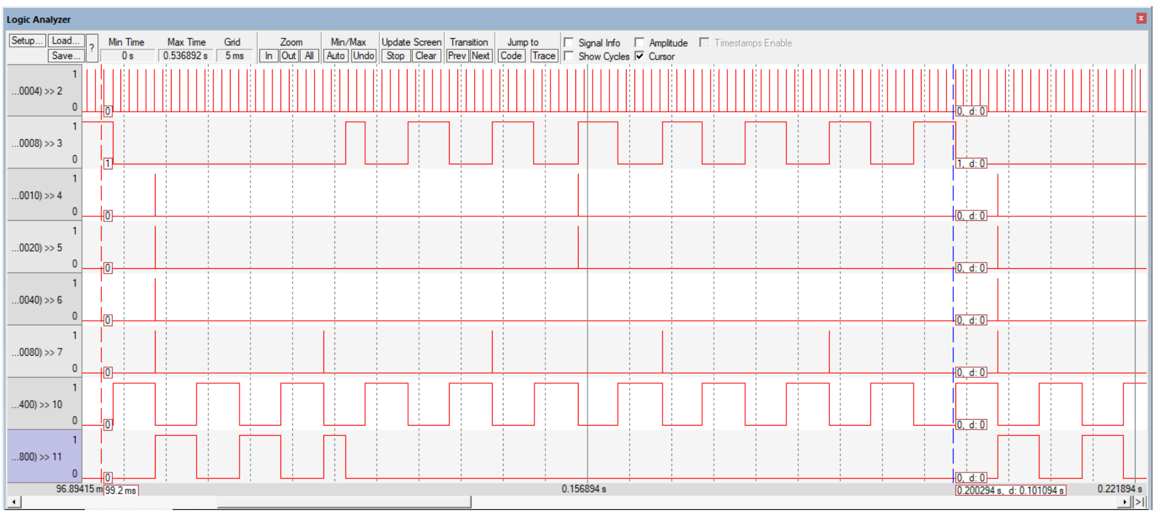


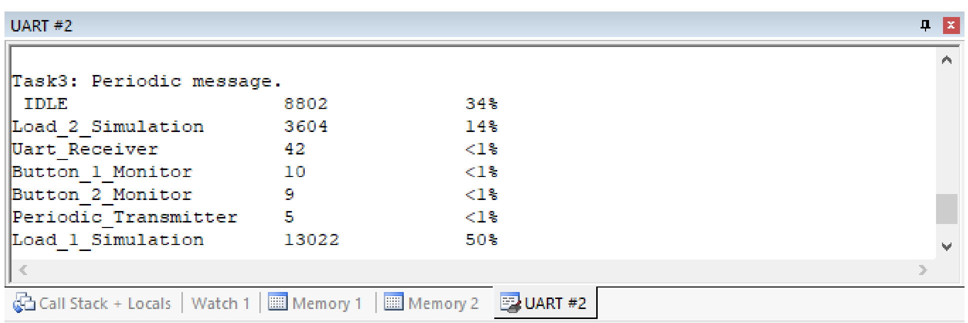


<u>Results:</u> SimSo simulation shows that the system is schedulable.

Keil Simulation

Port0.Pin0	Task1Button	Port0.Pin5	Task2
Port0.Pin1	Task2Button	Port0.Pin6	Task3
Port0.Pin2	Tick	Port0.Pin7	Task4
Port0.Pin3	IDLE	Port0.Pin10	Task5
Port0.Pin4	Task1	Port0.Pin11	Task6





<u>**Results:**</u> Keil simulation results shows that the system is schedulable with a hyper-period of 100 ms and repeating afterwards.

Conclusion

For the given system and the requirement of using EDF algorithm for the scheduler. According to the analytical method the system is schedulable and that is confirmed by SimSo software simulation and keil software simulation of the LPC2129 µcontroller.

References

Enrico Carraro (2016), Implementation and Test of EDF and LLREF Schedulers in FreeRTOS.

FreeRTOS (2022), Coding Standard, Testing and Style Guide.