Earliest Deadline Scheduler
(EDF)

Implementation of EDF in FreeRTOS

Mustafa Mahmoud Ali

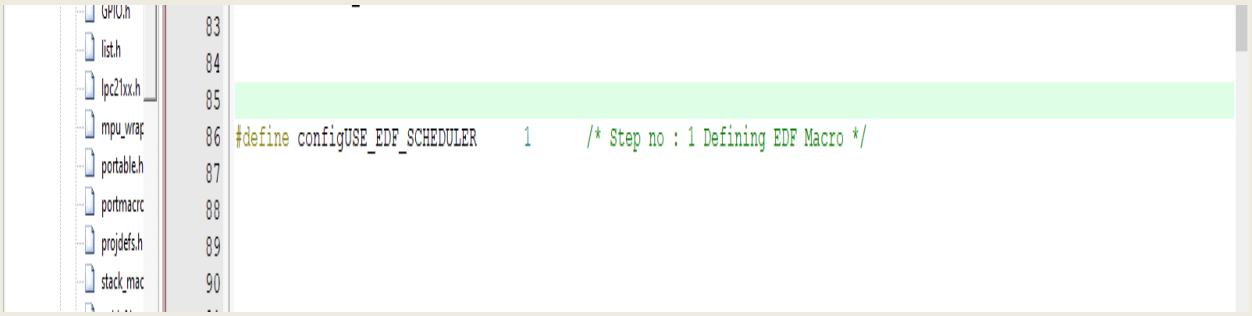
Earliest Deadline First:

The first scheduler we will implement is based on the 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.

Implementation:

Step one:

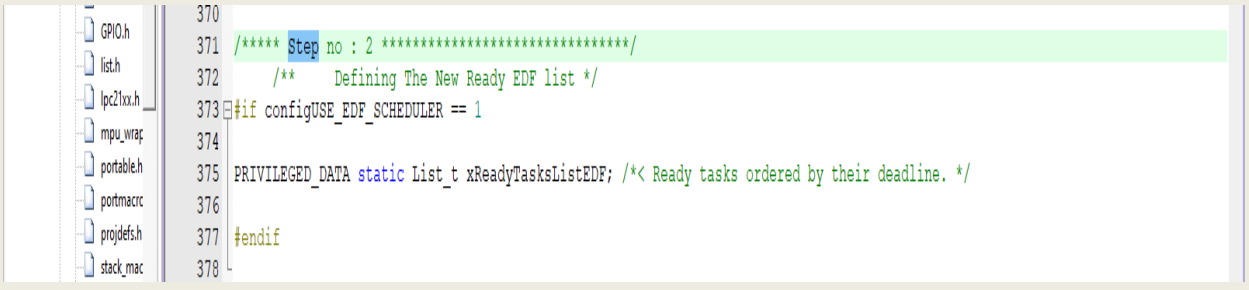
Defining the EDF Scheduler Macro to the “FreeRTOSConfig.h” file



And the rest of the steps in “Task.c” File.

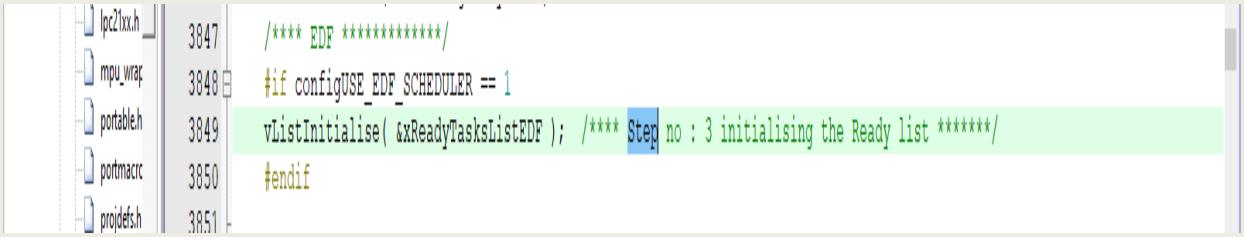
Step two:

Ready List is declared: *xReadyT asksListEDF*



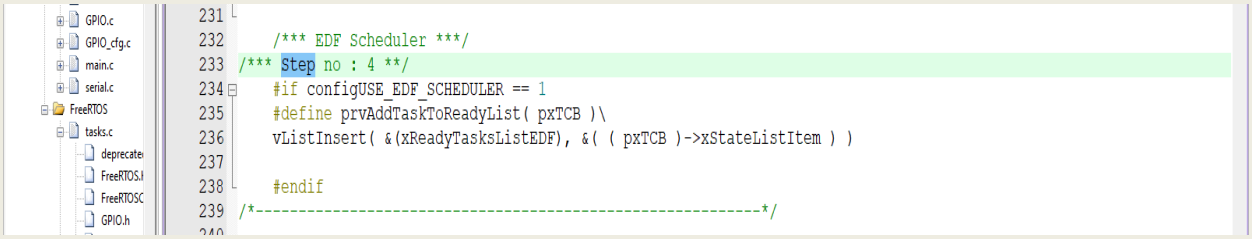
Step three:

prvInitialiseT askLists() method, that initialize all the task lists at the creation of the first task, is modified adding the initialization of *xReadyT asksListEDF* :



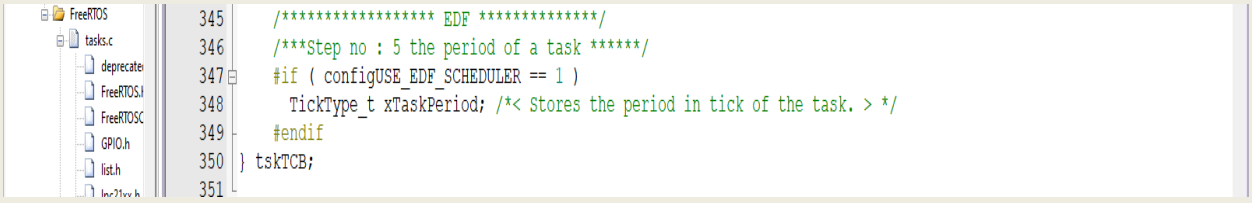
Step four:

prvAddT askT oReadyList() method that adds a task to the Ready List is then modified as
follows:



Step five:

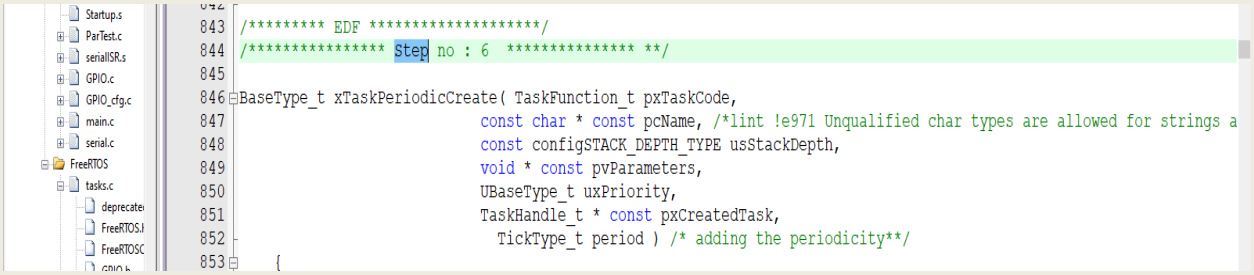
A new variable is added in
the tskTaskControlBlock structure (TCB):



Step six:

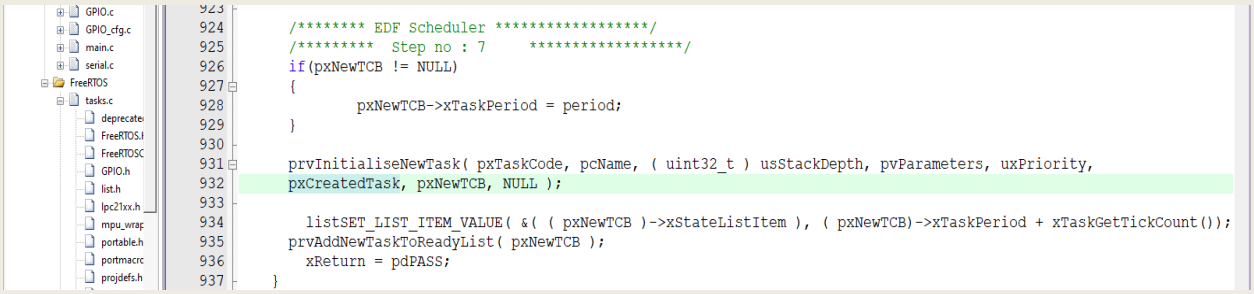
Modifying the xCreatTask to be xperiodicCreate() with new features which are :

adding the new period and update it before adding it to the xStatelist



```
/********* EDF ********************/
/**************** Step no : 6  *************** **/

BaseType_t xTaskPeriodicCreate( TaskFunction_t pxTaskCode,
                                const char * const pcName, /*lint !e971 Unqualified char types are allowed for strings a
                                const configSTACK_DEPTH_TYPE usStackDepth,
                                void * const pvParameters,
                                UBaseType_t uxPriority,
                                TaskHandle_t * const pxCreatedTask,
                                  TickType_t period ) /* adding the periodicity**/
    {
```

Step seven:

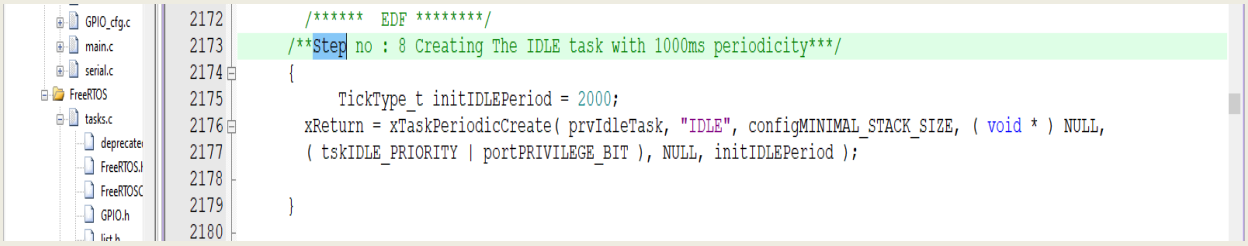


```
    /******** EDF Scheduler ******************/
    /*********  Step no : 7     ******************/
    if(pxNewTCB != NULL)
    {
            pxNewTCB->xTaskPeriod = period;
    }

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

      listSET_LIST_ITEM_VALUE( &( ( pxNewTCB )->xStateListItem ), ( pxNewTCB)->xTaskPeriod + xTaskGetTickCount());
    prvAddNewTaskToReadyList( pxNewTCB );
      xReturn = pdPASS;
  }
```

Step eight:

Modifying the IDLE task to be longest task to avoid being in running state while

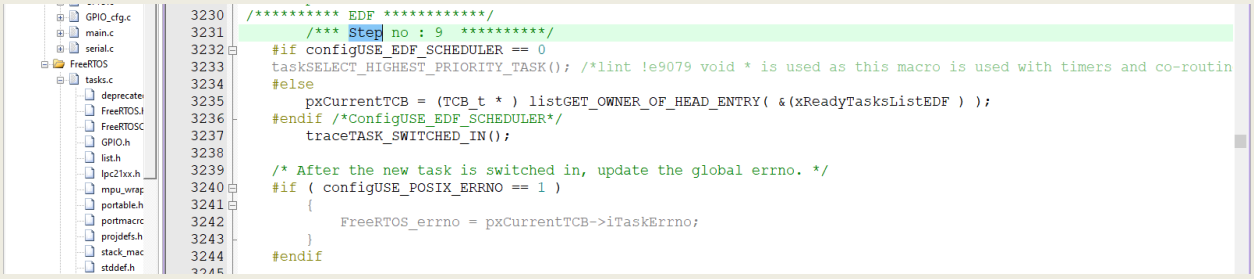
Other task must be in instead



```
      /******  EDF ********/
    /**Step no : 8 Creating The IDLE task with 1000ms periodicity***/
    {
        TickType_t initIDLEPeriod = 2000;
      xReturn = xTaskPeriodicCreate( prvIdleTask, "IDLE", configMINIMAL_STACK_SIZE, ( void * ) NULL,
      ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), NULL, initIDLEPeriod );

    }
```

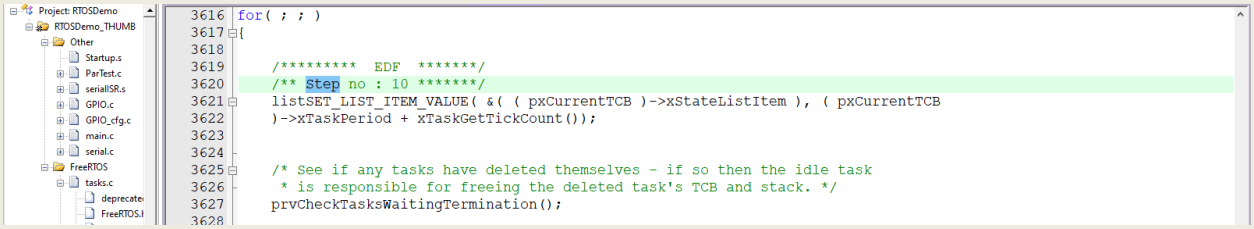
Step nine:

Make the TCB always points to the top of the ready list which is ready to run



```
3230  /********** EDF ************/
3231        /*** Step no : 9  **********/
3232      #if configUSE_EDF_SCHEDULER == 0
3233      taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-routin
3234      #else
3235          pxCurrentTCB = (TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &(xReadyTasksListEDF ) );
3236      #endif /*ConfigUSE_EDF_SCHEDULER*/
3237          traceTASK_SWITCHED_IN();
3238
3239      /* After the new task is switched in, update the global errno. */
3240      #if ( configUSE_POSIX_ERRNO == 1 )
3241          {
3242              FreeRTOS_errno = pxCurrentTCB->iTaskErrno;
3243          }
3244      #endif
```

Step ten:



```
3616  for( ; ; )
3617  {
3618
3619      /*********  EDF  *******/
3620      /** Step no : 10 *******/
3621      listSET_LIST_ITEM_VALUE( &( ( pxCurrentTCB )->xStateListItem ), ( pxCurrentTCB
3622      )->xTaskPeriod + xTaskGetTickCount());
3623
3624
3625      /* See if any tasks have deleted themselves - if so then the idle task
3626       * is responsible for freeing the deleted task's TCB and stack. */
3627      prvCheckTasksWaitingTermination();
3628
```

-The Tasks in the main function are:

T1 {P: 5, E: 2.5, D: 5}, T2 {P: 15, E: 4.5, D: 15}, T3 {P: 20, E: 3.5, D: 20}

- First we need to calculate the U for the System (CPU Load)

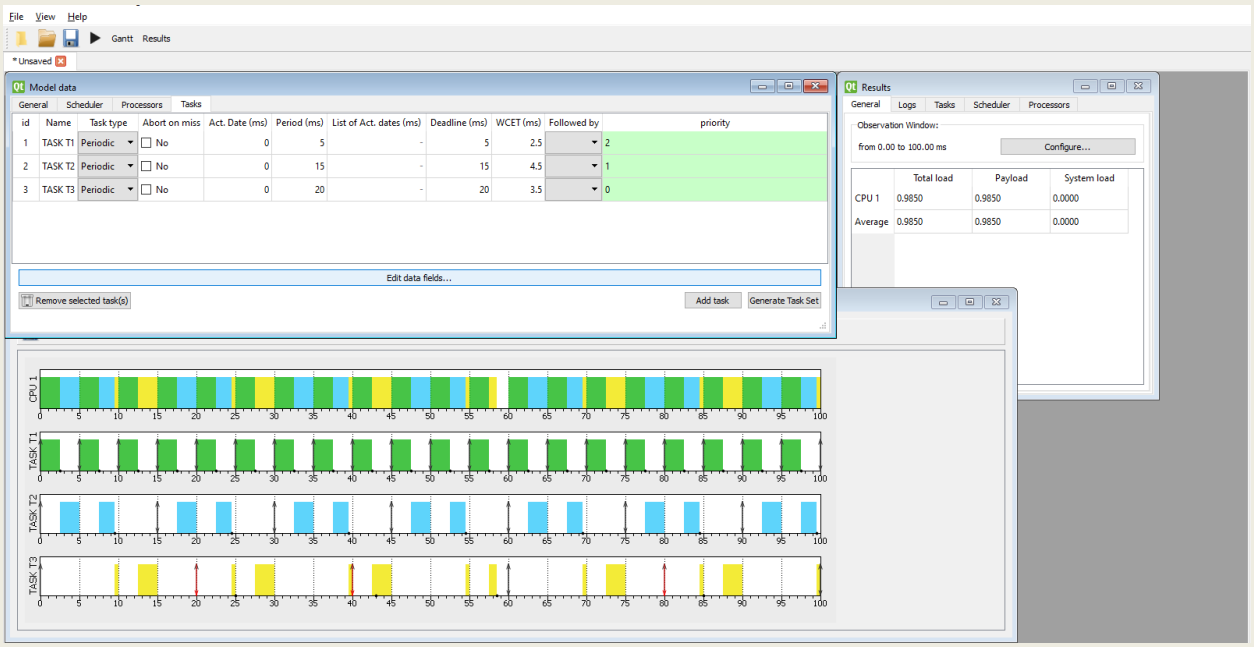
U = (2.5/5)+(4.5/15)+(3.5/20) = 0.975

- URM using the equation of RM = n(2^1/n – 1)

3(2^1/3 – 1) = 0.779

U > URM, Then the system is guaranteed not schedulable

Task 3 always misses its deadline time



(Rate-Monotonic Scheduler)

Calculating the Time Analysis For the system

For Task 1 :

W(1 to 5) = 2.5 + 0 = 2.5 < 5 T1 is schedulable

For Task 2 : The execution would happen after Task 1 finishes its time

W() = The execution of task 1 + task 2

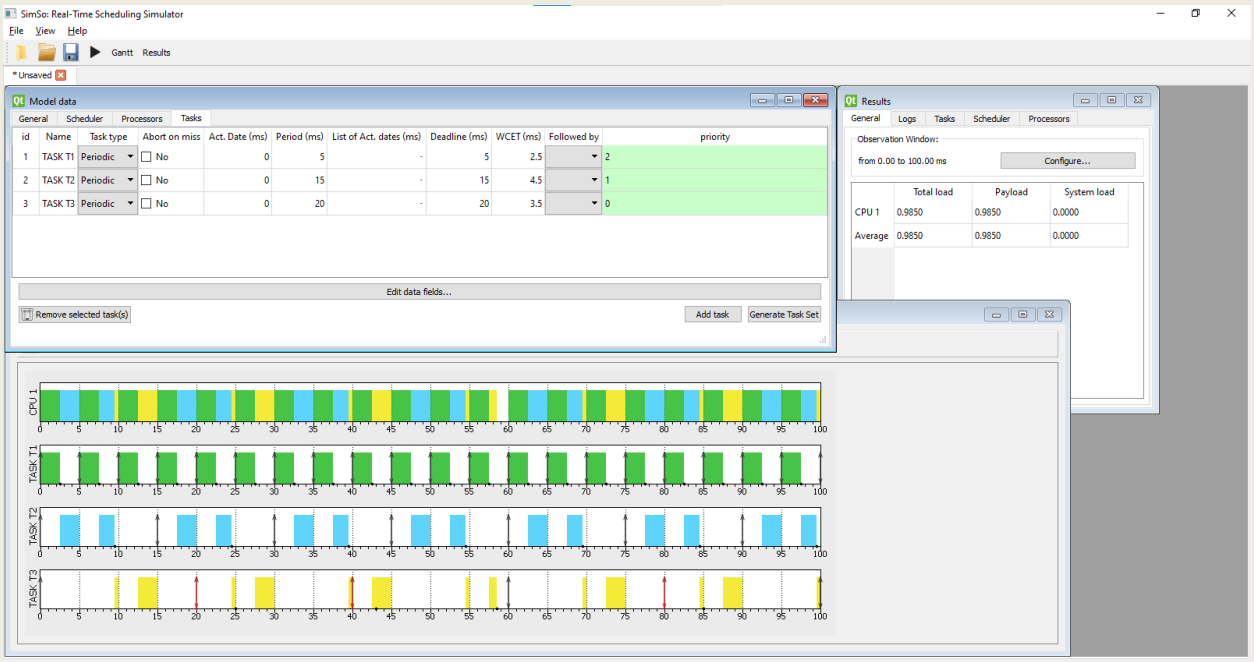
W(1 to 5) = 4.5 + (1/5)2.5 = 7

W(10) = 4.5 + (10/5)2.5 = 9.5

W(15) = 4.5 + (15/5)2.5 = 12 < 15 T2 is schedulable

For Task 3 : W() = Total execution time

W(16) = 3.5 + (16/5)4.5 + (16/5)2.5 = 25.9 > 20 Task 3 always misses its deadline



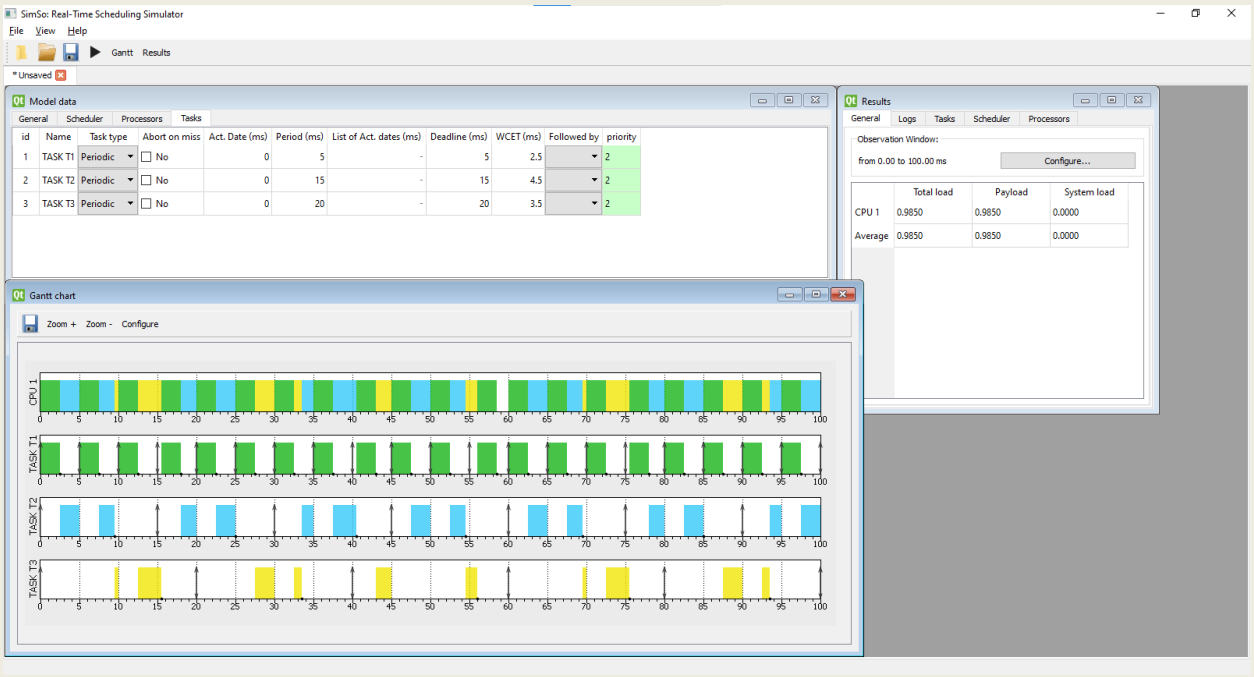
(Run-Time FP Scheduler)

In the EDF scheduler the system is said to be schedulable when

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

U = (2.5/5)+(4.5/15)+(3.5/20) = 0.975 < 1

Then the system is schedulable



Applying the System on the logic analyzer in keil: