**Scheduler’s Handbook**

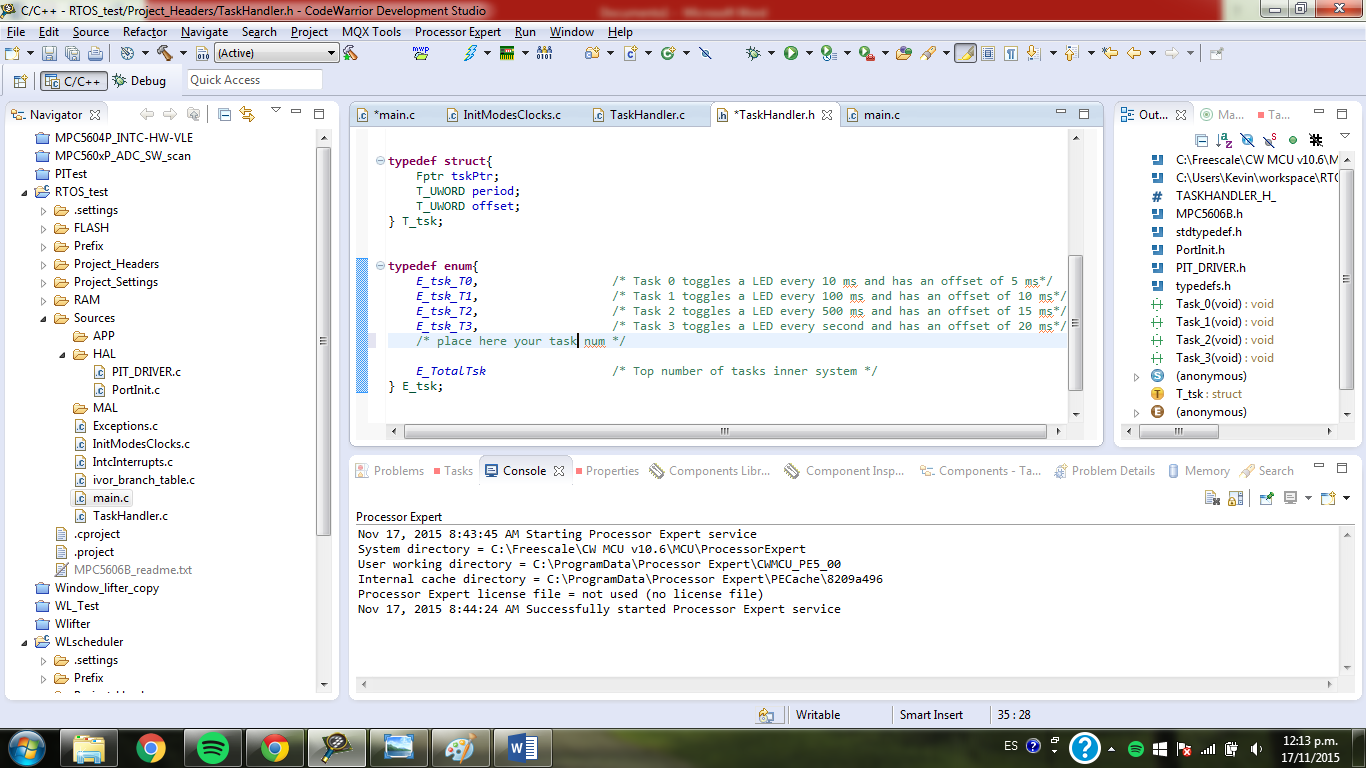
This scheduler is based on a not preemptive RTOS with a system responsiveness of 1 millisecond provided by a periodic interrupt timer. A Freescale TRK-MPC5606B is the system board and runs at 64 MHz on a 32 bit Power Architecture©.

According to its definition, tasks and feature design must meet the timing constraints depicted before and thus prevent deadline loss. Priority levels are inherent to task’s code order so the major priority tasks should be defined in the first lines in the task handler. To avoid system decay or system failure we recommend to add an offset timing for preventing worst case scenario, when multiple tasks request system resources.

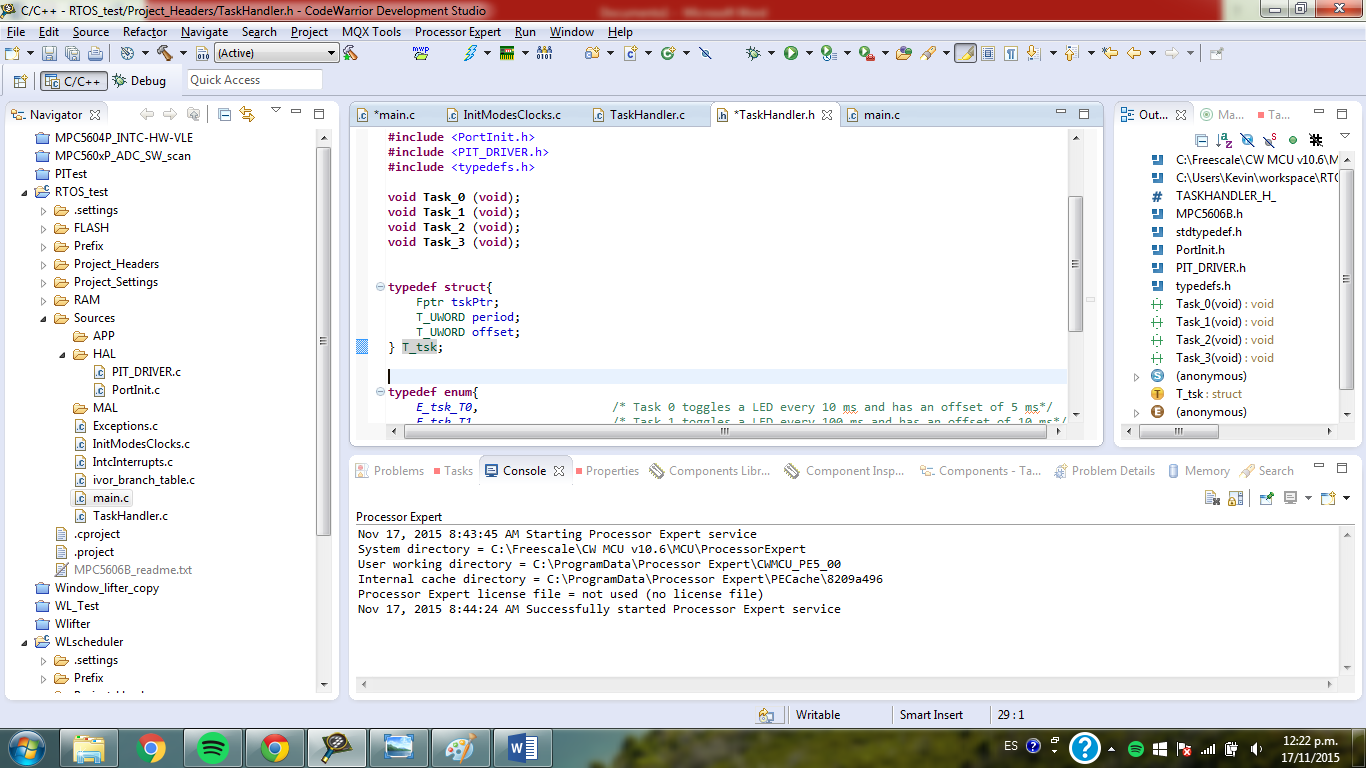
In order to add and implement a new task onto the system it is mandatory to give an overview of the main structure and operation of the scheduler. The scheduler performs a periodic interrupt every millisecond, which is the capability of the system to perform an action. This interrupt triggers a flag that let us evaluate whether or not a task has finished its execution and its period. In case a task, according to task state model, is in ready state and the last one executed has finished its period, the task will take over the system resources to perform their actions. This will continue up until the period from the task runs out and then another task will be executed then. The tasks will be executing one by one accomplishing a continuous loop.

***Adding a new task:***

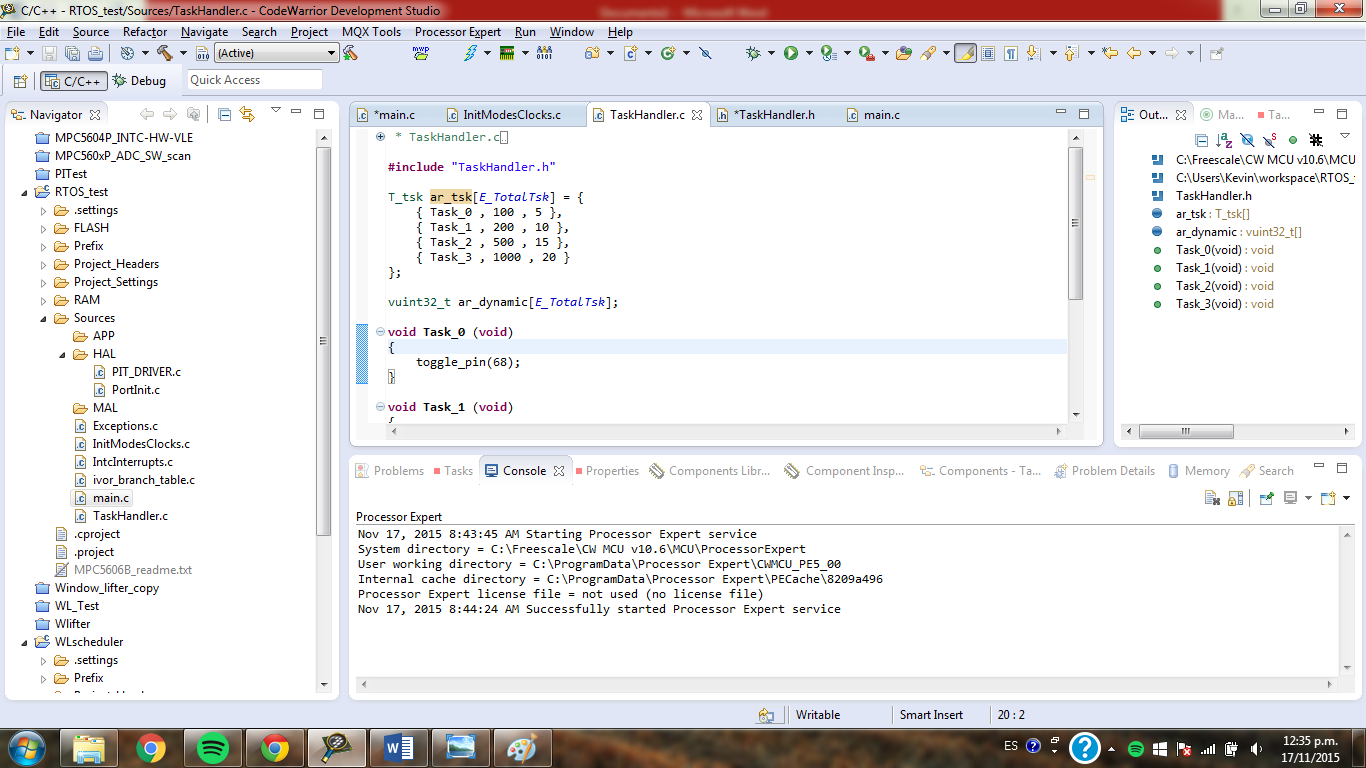
1. On the project header TaskHandler.h [Project\_Headers\MAL\TaskHandler.h] locate the E\_tsk enum type and add your own value noticing that the last value given *E\_TotalTsk* represents the total number of tasks in the system. Add the relevant comments for hints of your task number.



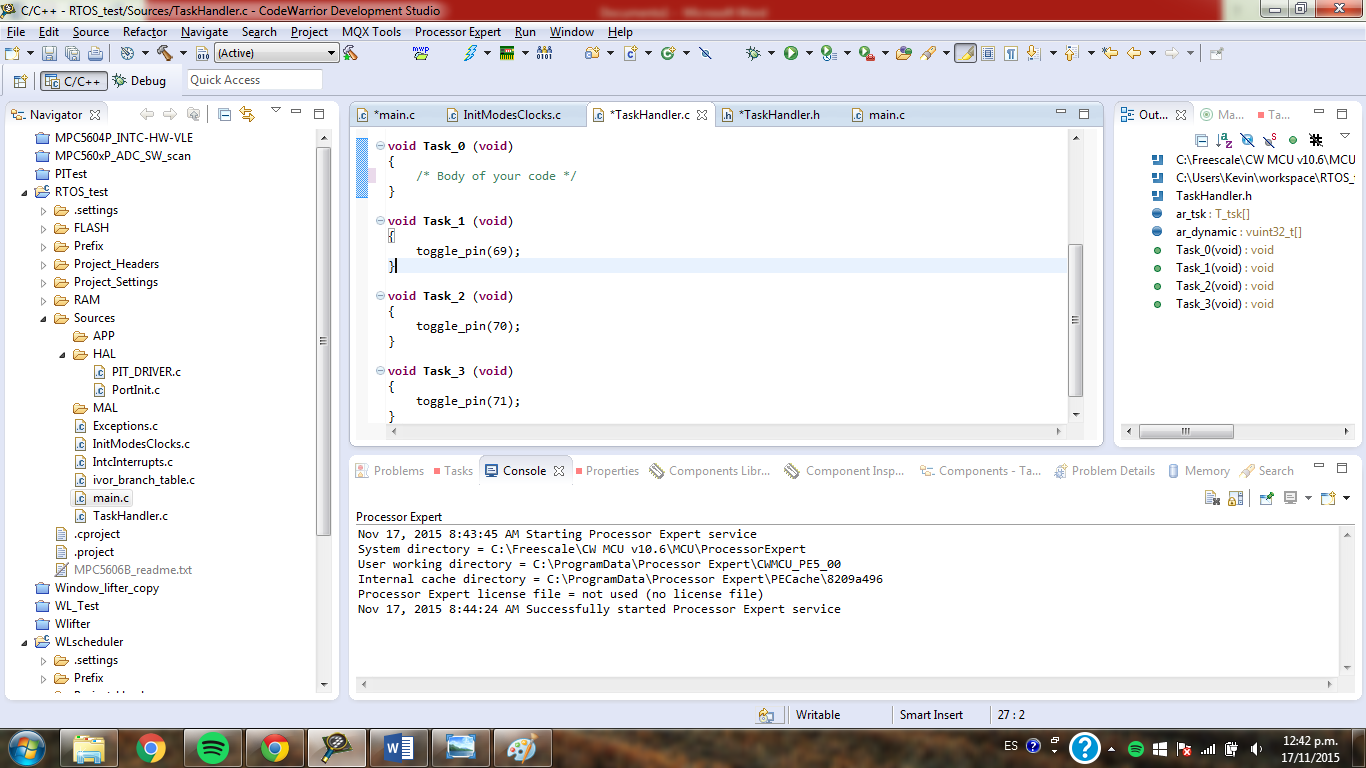
1. There is a task structure type T\_tsk that holds 3 members: a pointer to the task to perform, a value that stores the period of current task and an offset value to prevent worst case scenario.



This structure is initialized in an array called ar\_tsk on TaskHandler.c [Project\_Headers\MAL\TaskHandler.c] where you must add the name of the function that refers to the task and its components by design and type the comments to hint your code and the task.



1. On the TaskHandler.c [Project\_Headers\MAL\TaskHandler.c] file your tasks must be coded and designed in functions that doesn’t return and admits any value as depicted below:



Do not forget to place the function prototype in TaskHandler.c [Project\_Headers\MAL\TaskHandler.c] to make the code useful.

1. Finished. The task implementation must run over the system with the prescriptions defined by the user.