

RTOS:

Non-preemptive scheduler manual

**Guillermo Ramírez**

**Óscar Francisco Miranda García**

Automotive Entry Program 2015

Coach: Carlos Calvillo

**Continental México**

Campus: Santa Anita

Date: November 16th, 2015

**Introduction**

This scheduler is a non-preemptive task delay depending. It’s been developed for simple projects and for soft real systems. It was implemented in a Freescale MPC5606B development board. It provides the user to add as much tasks as necessary and the configuration of the delays and the clock tick, etc.

**Features**

This manual includes and provides:

* The general scheduler code
* The capability to add several tasks,
* modify the task’s period and offset,
* and establish the clock tick.
* An example of a program with four tasks that control the frequency of four LED’s.

**Fundamentals**

***-Clock Tick***

The clock tick is set in the timer module (STM) of the hardware that runs at 16 MHz (this value can be modified) and an ISR function is used.

To establish the clock tick use the CMP register of the STM module using the following formula:

Were Clock Tick Hex is a hexadecimal value, Clock Speed Hex is the hexadecimal magnitude of the clock speed, and Desired Clock Tick Time is the value in seconds of the Clock Time.

To keep the control of the clock tick a header OS\_Init.h is used. It has a variable that is used to mark the deadline of the tick and the interrupt function **Tick\_Flag()** that clears the **ub\_TickFlag** , the clock interrupt flag of the STM, and reset the CNT register. This function is showed in the OS\_Init.c file of the next page.

OS\_Init.h

**T\_UBYTE ub\_TickFlag;**

**void** **Tick\_Flag**(**void**);

OS\_Init.c

**void** **Tick\_Flag**(**void**)

{ /\* --------------------------------------------------------------------------

\* Name : Tick\_Flag

\* Description : Check if the channel 0 of the STM as reached 10ms

\* and raise a flag when reached

\* Parameters : void

\* Return : void

\* -------------------------------------------------------------------------

\*/

**if** (STM.CH[0].CIR.B.CIF)

{

ub\_TickFlag = 1; /\* Clear tick clock flag \*/

STM.CH[0].CIR.B.CIF = 1; /\* Clear interrupt flag \*/

STM.CNT.R = 0; /\*Reset counter\*/

}

}

***-Tasks***

To add a task:

In the TASK\_Init.h add as much enums as necessary inside **E\_TASK** enum, the most important constant is **E\_TASK\_NUM** that tells the number of the total tasks of the scheduler.

Each task is defined by the following:

A structure S\_TASK has three member a handler called **rp\_Tasks** to call the task, a 32 bit **rul\_Period**, which is used to define the period of such task, and a 32 bit **rul\_Offset**, which is used to define the task’s offset.

The tasks are declared and defined in both, the TASK\_Init.h and Task\_Init.c files:

TASK\_Init.h

**typedef** **struct** {

**void**(\* rp\_Tasks)(**void**); /\*Pointer that'll call every task\*/

T\_ULONG rul\_Period; /\*Period of task\*/

T\_ULONG rul\_Offset; /\*Offset of task\*/

}S\_TASK;

**typedef** **enum**{

*E\_TASK1*,

*E\_TASK2*,

… …

*E\_TASK\_NUM*

}E\_TASK;

In the TASK\_Init.c an array of four **S\_TASK** variables must be used to define the tasks, and to define the periods and offsets the format will be the following:

TASK\_Init.c

**#define** taskPeriod1 XXXX

… …

**#define** taskPeriodN XXXX

**#define** taskOffset1 XXXX

… …

**#define** taskOffsetN XXXX

**const** S\_TASK taskName[*E\_TASK\_NUM*] = {

{taskName1, taskPeriod1, taskOffset1 },

{taskName2, taskPeriod2, taskOffset2 },

{taskName3, taskPeriod3, taskOffset3 },

… … … …

{taskNameN, taskPeriodN, taskOffsetN }

};

Note that **E\_TASK\_NUM** represents the number of tasks, in this case, four tasks, and that the period and the offset are numeric values.

***-Use of the scheduler***

The scheduler controls the tasks with delays that are defined by the clock tick, in other words, uses the round-robin algorithm. To have a good performance of each task, a task must be small enough to fit within the clock tick. If a task is larger than the clock tick, it would be pause and will be dispatched in the next clock tick if other tasks are waiting. A task offset is necessary to avoid executing tasks at the same time, but this only reduces the possibility of such case, it doesn’t eliminate it.