# Scheduler Report

Author: Habib Alejandro Apez González

Date: 2017/11/17

**Previous Comments**

The Scheduler uses the Binary Progression Algorithm to determine the execution of the tasks.

1. The scheduler was implemented in the S32K144EVB.
2. This project was compiled in IAR.
3. The code was developed following the AUTOSAR Software Architecture: MCAL, HAL, SERVICES and APP Layers were defined.
4. To implement the Scheduler's Tick, the SysTick Interruption of the microcontroller was used. The Scheduler has a 781.25us Tick.
5. To implement the delays, the LPIT0 was used.
6. All code files of the project are attached as well as the Scheduler Design document.
7. The hardware of the window lifter project was used for this project.

## 1. Tasks Executions

All tasks execute the same code and have the same execution time. The following image shows the definition of *SchM3p125ms()*, this definition is provided to the rest of the tasks too. In the Task, a LED of a segment bar is turned on and kept on during 500 us, then it is turned off. The delay time is 500 us because the task execution time must be shorter than the Tick.

```
/************************************************************
 *  Name                 : SchM_12p5ms_Task
 *  Description          : Executes a task each 12.5ms
 *  Parameters           : [void]
 *  Return               : void
 *  Critical/explanation : No
 ************************************************************/
void SchM_12p5ms_Task(void){  /* Code Task2*/
  segmentbar_TurnOnSeg2();
  delays_Wait500us();          /* Comment this line to observe the CPU load change */
  segmentbar_TurnOffSeg2();
}
```

Note that a task execution time longer than the Tick would cause a collision and lead the Scheduler to the Overload State.

## 2. Tasks Periods

The Task periods are:

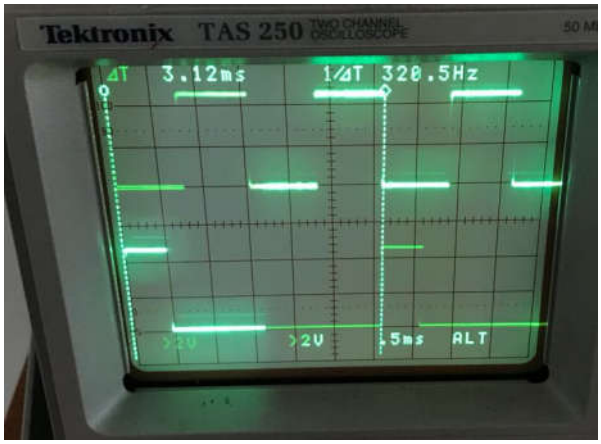
| TASK_NUMBER | TASK_ID | Minumun Executions | Offset | Mask | Task period (ms) | Number of executions |
|---|---|---|---|---|---|---|
| 1 | TASK_3P125MS | 96 | 0 | 3 | 3.125 | 96 |
| 2 | TASK_6P25MS | 48 | 1 | 7 | 6.250 | 48 |
| 3 | TASK_12P5MS | 24 | 2 | 15 | 12.500 | 24 |
| 4 | TASK_25MS | 12 | 3 | 31 | 25.000 | 12 |
| 5 | TASK_50MS | 6 | 5 | 63 | 50.000 | 6 |
| 6 | TASK_100MS | 3 | 6 | 127 | 100.000 | 3 |

To check the Tasks periods, an analogic oscilloscope was used. The following images show the period of each task. At the top of the oscilloscope's screen is the Tick signal and at the bottom is the Task signal.

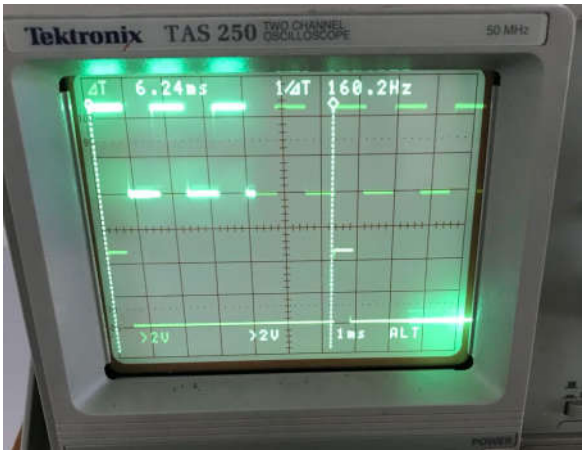
The Tick signal is generated toggling a pin (blue LED of the development board) in the interrupt function that calls to the callback of *SchM_OsTick()*. In this case, that interrupt function is SysTick_*Handler()*. Because of that, the Tick signal is a square signal. Each edge (rising and falling) represents an increment in the scheduler counter *SchM_OsTickCounter*.

```
/*============================================================================*/
/*************************************************************
 *  Name                 : SystTick interruption
 *  Description          : Moves the Window upwards
 *  Parameters           : [void]
 *  Return               : void
 *  Critical/explanation : No
 *************************************************************/
void SysTick_Handler(void){
  if ( NULL!= GlbSysTickCallback)
          GlbSysTickCallback();
  leds_ToggleBlueBoardLED();
}
```

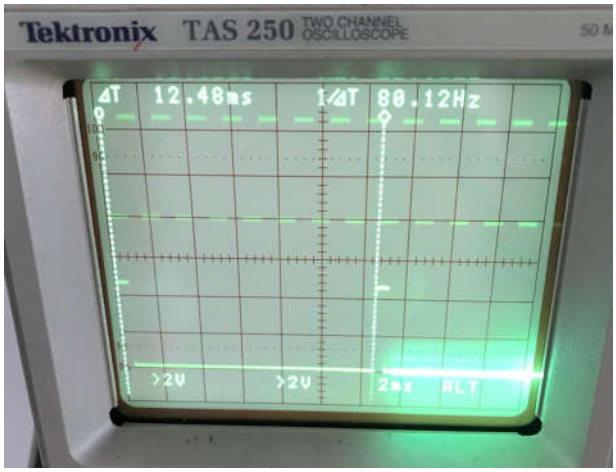
- *SchM3p125ms*



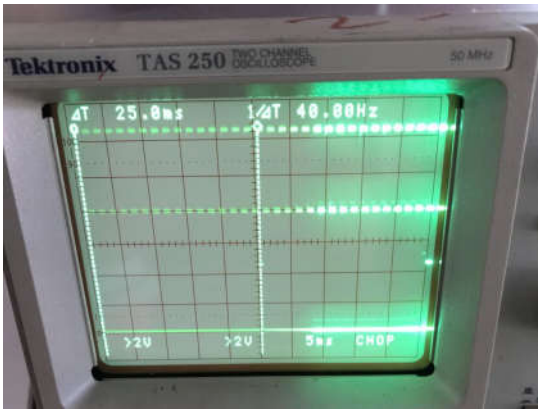

- *SchM6p25ms*



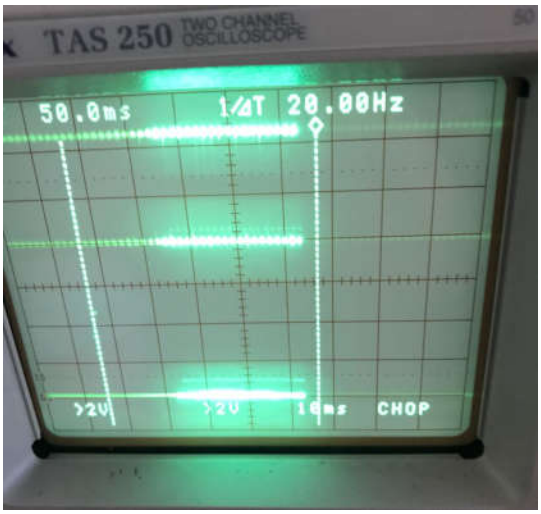

- *SchM12p5ms*



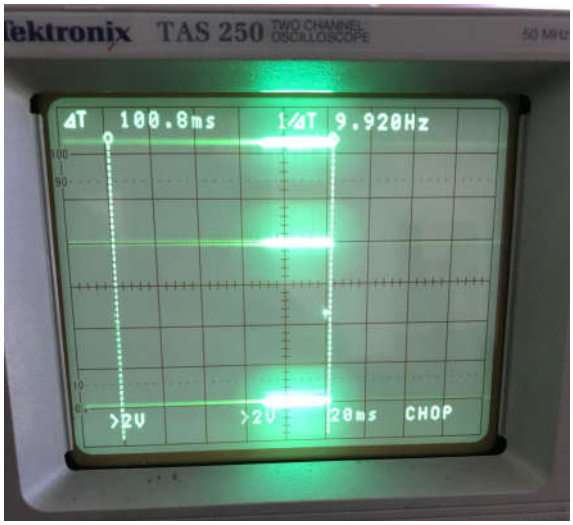
- *SchM25ms*




- *SchM50ms*




- *SchM100ms*

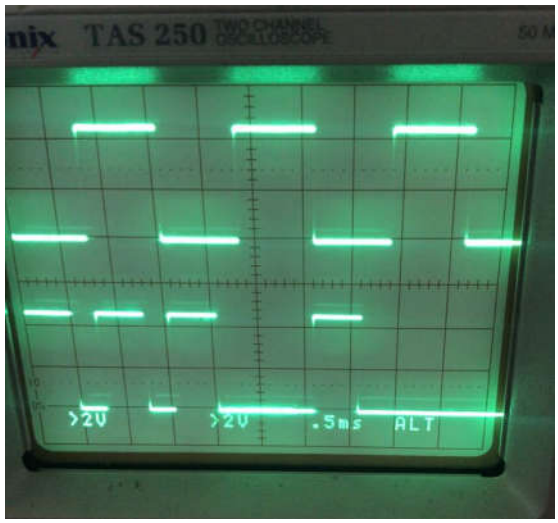


## 3. CPU Load

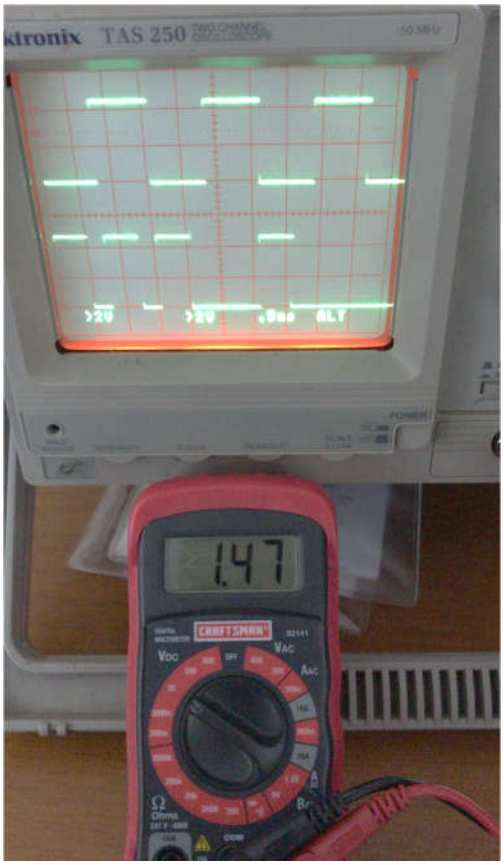
To measure the CPU load, a LED (blue LED ) was turned on before the Schedules executes a Task and turned off when the Task has finished its execution.

### a. Normal Operation

The following image shows the CPU load in normal operation. At the top of the oscilloscope's screen is the Tick signal and at the bottom is the CPU load signal.

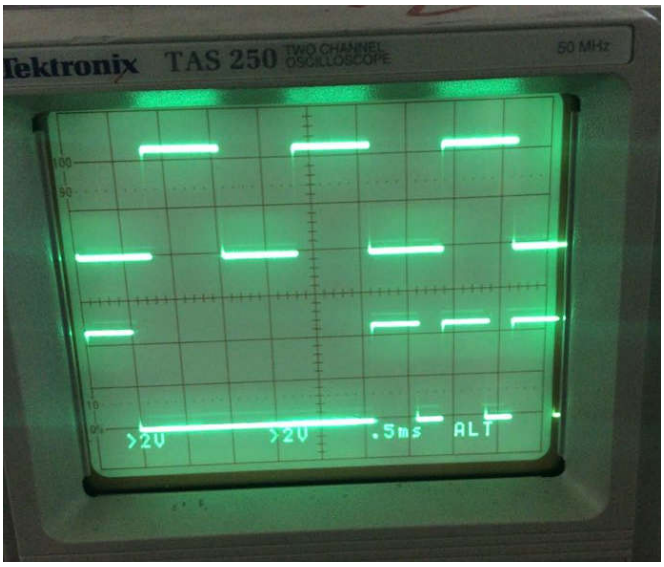



To a have numerical representation of the CPU load, the voltage at the pin of the LED was measured using a multimeter. So, if the CPU load is maximum (50%), the voltage will be close to 5v. On the other hand, if the CPU load is 50%, the voltage will be close to 2.5V, similar to the PWM. The Following image shows the voltage in the pin of the LED in normal conditions.

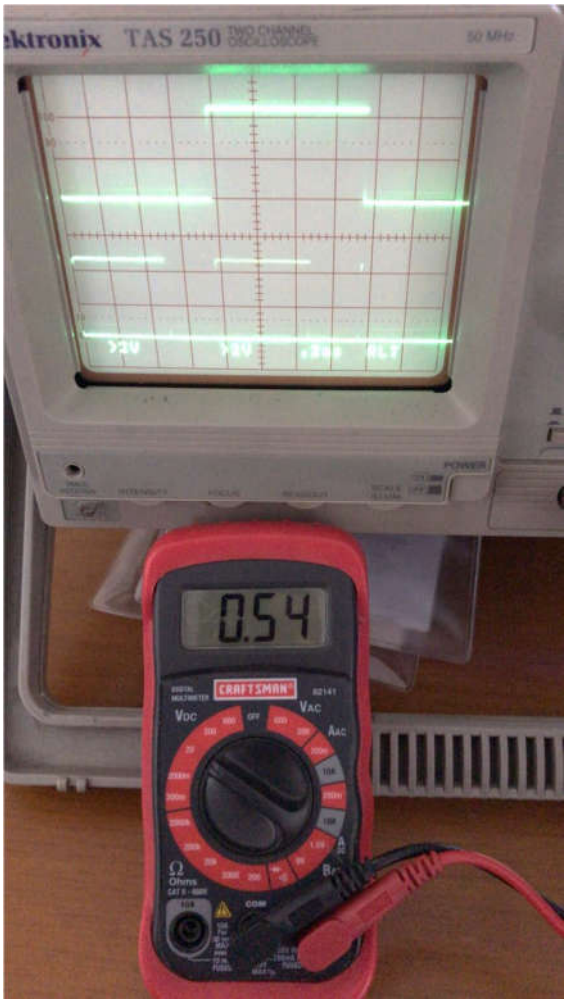



### b. Reduced CPU load Operation

To reduce the CPU load, the delays in *SchM_3p125ms_Task()* and *SchM_12p5ms_Task()* were removed. In his way, the CPU load decreases as shown in the following image.



The following image shows that the voltage at the pin of the LED also decreases.




Note that a longer execution task time in a task with a short period of time, will have a greater effect on the CPU load than a longer execution task time in a task with a longer period of time.

## 4. Scheduler Overload

To overload the CPU, *delays_Wait500us()* was called twice in order to have execution task time which is longer than the Tick time and force the Overload condition in the scheduler When the Scheduler is overloaded, the status of the scheduler is set to overload and the green LED is turned on.

The Overloard condition is reached when the Scheduler is running a task and another task is in ready state. The following figure shows the code with the Overload detection.

```
/*************************************************************
 *  Name                 : SchM_OsTick
 *  Description          : OS Tick Function
 *  Parameters           : [void]
 *  Return               : void
 *  Critical/explanation : No
 *************************************************************/
void SchM_OsTick(void){
  T_UBYTE lub_Index;

    for(lub_Index = 0; lub_Index < SchM_ConfigGlobal->SchM_NumOfTasks; lub_Index++){
      if(((SchM_ConfigGlobal->SchM_TaskDescriptor[lub_Index].SchM_TaskMask) & (SchM_Control.SchM_OsTickCounter)
        if(SCHM_RUNNING == SchM_Control.SchM_State){
          SchM_Control.SchM_State = SCHM_OVERLOAD;      // Set Overload Flag
          leds_TurnOnDownLED();
        }
        SchM_TaskControlBlock[lub_Index].SchM_TaskState = SCHM_TASK_STATE_READY;
      }
      else {
        // No-Task to be executed
      }
    }
  SchM_Control.SchM_OsTickCounter++;
 }
```

The following image shows the system when the Scheduler is overloaded.