|  |
| --- |
| **Software Test Plan** |

|  |  |
| --- | --- |
| Nr. : | 01 |
| Title: | OS Scheduler |

**Contents**

1. Test Specification Information 3

2. Module Test Cases 3

3. Integration Test Cases 3

4. Naming rules for functions 3

# Test Specification Information

|  |  |  |
| --- | --- | --- |
| **Date of issue (MM/DD/YY)** | **Test Developer** | **Revision & Description** |
| 11/03/2014 | Esteban, Miguel | 1.0 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Module Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.0** |  |
| **Requirements covered** | | |
| 1.1 | | |
| **Test Procedure** | | |
| Scheduler initialization shall be supported through SchM\_Init API | | |
| **Expected Results** | | |
| Initialization of Scheduler via SchM\_Init. | | |
| **Actual Results** | | **Test Results** |
| Scheduler is initialized by SchM\_Init API | | PASS |
| **Comments** | | |
| Parameters are not supported currently since there not stack allocation. See Figure 1 | | |

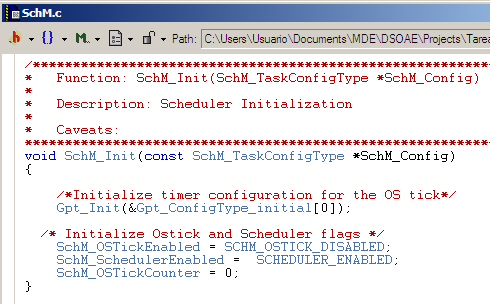


Figure 1

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.1** |  |
| **Requirements covered** | | |
| 1.2 | | |
| **Test Procedure** | | |
| Scheduler De-Initialization shall be supported through SchM\_DeInit API | | |
| **Expected Results** | | |
| De-Initialization of Scheduler via SchM\_DeInit | | |
| **Actual Results** | | **Test Results** |
| Scheduler is De-Initialized by SchM\_DeInit API | | PASS |
| **Comments** | | |
| Only De-Initialization Scheduler flag is updated since there are not resources to de-initialize. See Figure 2 | | |

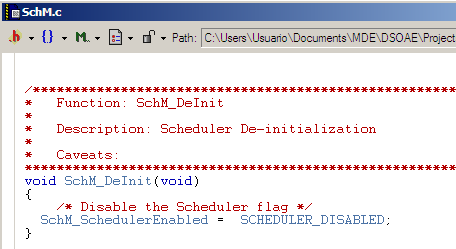


Figure 2

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.2** |  |
| **Requirements covered** | | |
| 1.3 | | |
| **Test Procedure** | | |
| Scheduler Start shall be supported through SchM\_Start API | | |
| **Expected Results** | | |
| Start of the Scheduler via SchM\_Start API | | |
| **Actual Results** | | **Test Results** |
| Scheduler is started by SchM\_Start API | | PASS |
| **Comments** | | |
| See Figure 3. | | |

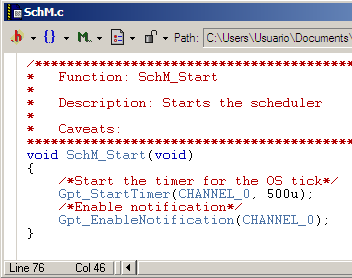


Figure 3

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.3** |  |
| **Requirements covered** | | |
| 1.4 | | |
| **Test Procedure** | | |
| OS Tick Callback shall be supported through SchM\_OsTick API | | |
| **Expected Results** | | |
| OS Tick Callback support via SchM\_OsTick API | | |
| **Actual Results** | | **Test Results** |
| OS Tick Callback is supported by SchM\_OsTick API | | PASS |
| **Comments** | | |
| See Figure 4 | | |

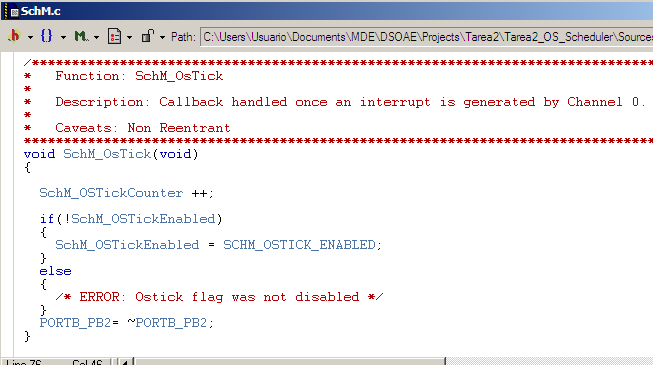


Figure 4

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.4** |  |
| **Requirements covered** | | |
| 1.5 | | |
| **Test Procedure** | | |
| Background task shall be supported through SchM\_Background API | | |
| **Expected Results** | | |
| Background task support via SchM\_Background API | | |
| **Actual Results** | | **Test Results** |
| Background task is supported by SchM\_Background API | | PASS |
| **Comments** | | |
| See Figure 5 | | |

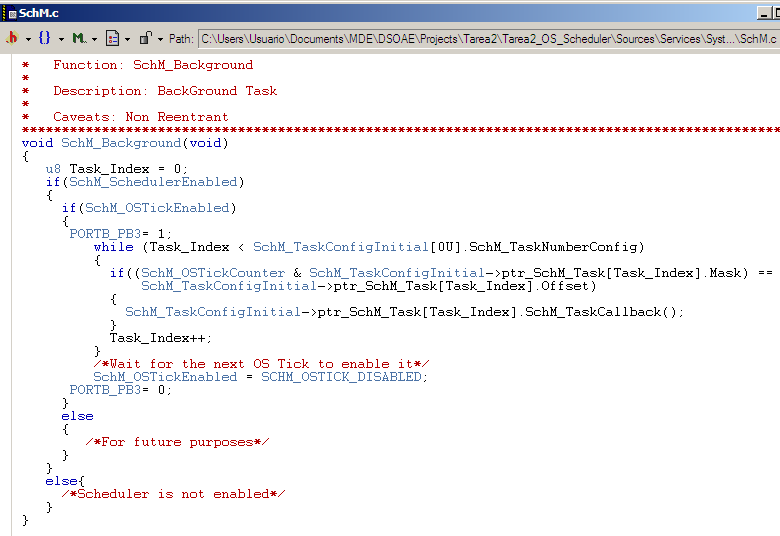


Figure 5

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.5** |  |
| **Requirements covered** | | |
| 1.6 | | |
| **Test Procedure** | | |
| Callback functions shall be referred as per the task period:  SchM\_Task\_##period(void) E.g. SchM\_Task\_1p56ms(void) | | |
| **Expected Results** | | |
| Callback functions referred as per the task period | | |
| **Actual Results** | | **Test Results** |
| Callback functions are referred as per the task period | | PASS |
| **Comments** | | |
| See Figure 6 | | |



Figure 6

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.6** |  |
| **Requirements covered** | | |
| 1.9 | | |
| **Test Procedure** | | |
| Scheduler Module Shall be allocated at BSW and Services layer from AUTOSAR | | |
| **Expected Results** | | |
| Scheduler module allocated at BSW and Services layer from AUTOSAR | | |
| **Actual Results** | | **Test Results** |
| Scheduler module is allocated at BSW and Services layer from AUTOSAR | | PASS |
| **Comments** | | |
| See Figure 7 | | |

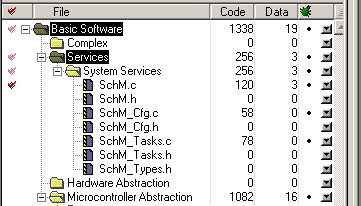


Figure 7

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.7** |  |
| **Requirements covered** | | |
| 1.11 | | |
| **Test Procedure** | | |
| SchM\_Cfg.c & SchM\_Cfg.h shall provide the task configuration table | | |
| **Expected Results** | | |
| Task configuration table provided in SchM\_Cfg.c & SchM\_Cfg.h | | |
| **Actual Results** | | **Test Results** |
| Task configuration table is included in SchM\_Cfg.c & SchM\_Cfg.h | | PASS |
| **Comments** | | |
| See Figure 8 & Figure 9 | | |

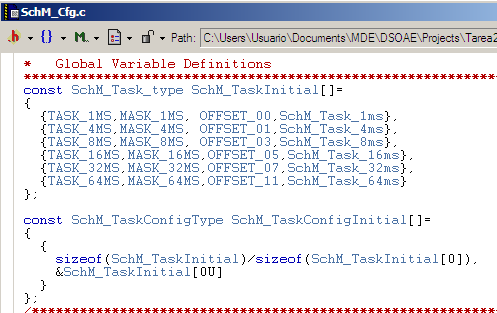


Figure 8

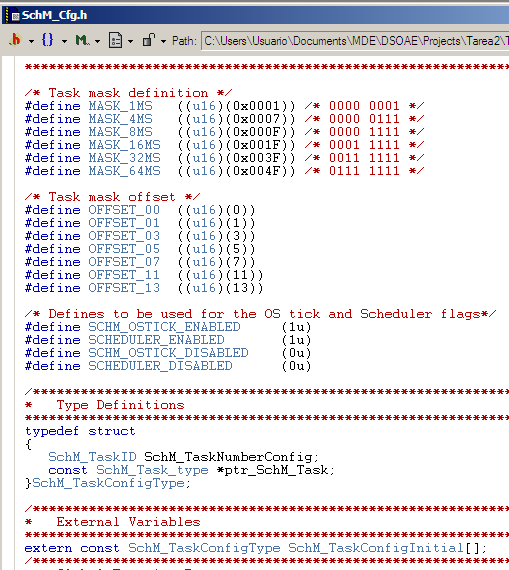


Figure 9

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.8** |  |
| **Requirements covered** | | |
| 1.12 | | |
| **Test Procedure** | | |
| SchM.c & SchM.h shall provide the main functionality of the Scheduler. OS-Tick callback shall be allocated in these files | | |
| **Expected Results** | | |
| Main functionality placed in SchM.c & SchM.h | | |
| **Actual Results** | | **Test Results** |
| Main functionality is allocated in SchM.c & SchM.h | | PASS |
| **Comments** | | |
| See Figure 10, Figure 11 & Figure 12 | | |

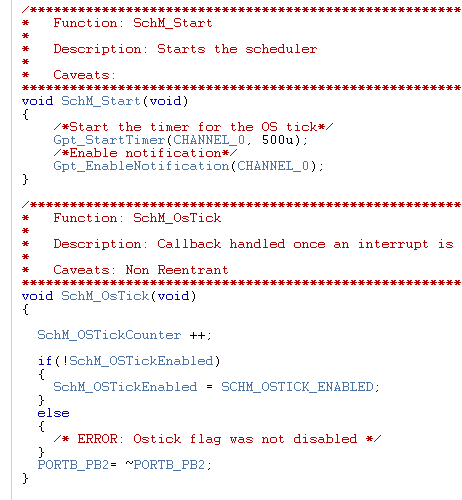
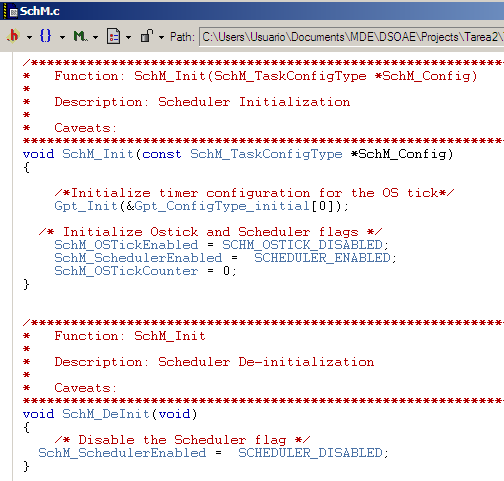


Figure 10



Figure 11

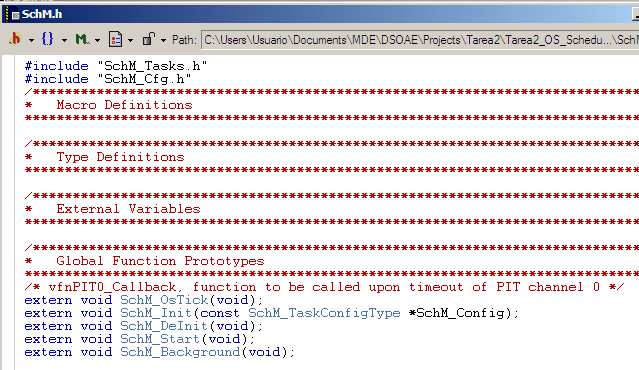


Figure 12

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.9** |  |
| **Requirements covered** | | |
| 1.13 | | |
| **Test Procedure** | | |
| SchM\_Types.h shall provide the Scheduler type definitions | | |
| **Expected Results** | | |
| Scheduler type definitions included in SchM\_Types.h | | |
| **Actual Results** | | **Test Results** |
| Scheduler type definitions are included in SchM\_Types.h | | PASS |
| **Comments** | | |
| See Figure 13 | | |

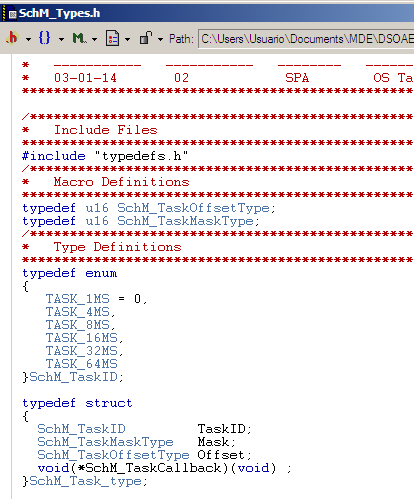


Figure 13

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.10** |  |
| **Requirements covered** | | |
| 1.14 | | |
| **Test Procedure** | | |
| SchM\_Tasks.c & SchM\_Tasks.h shall allocate the module's periodic tasks | | |
| **Expected Results** | | |
| Allocation of the periodic Tasks in SchM\_Tasks.c & SchM\_Tasks.h | | |
| **Actual Results** | | **Test Results** |
| Periodic Tasks are allocated in SchM\_Tasks.c & SchM\_Tasks.h | | PASS |
| **Comments** | | |
| See Figure 14, Figure 15 & Figure 16 | | |

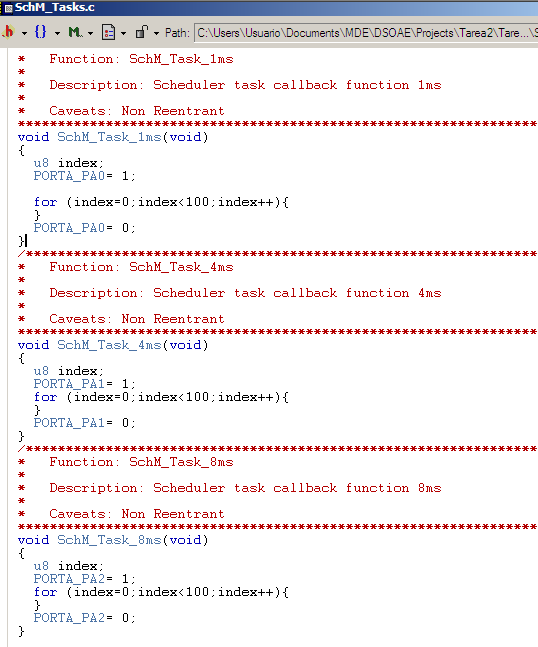


Figure 14

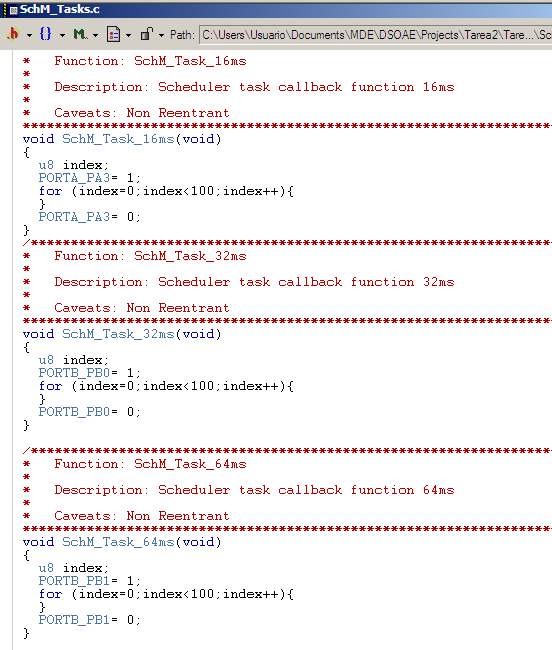


Figure 15

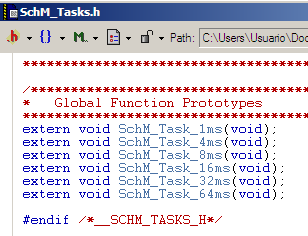
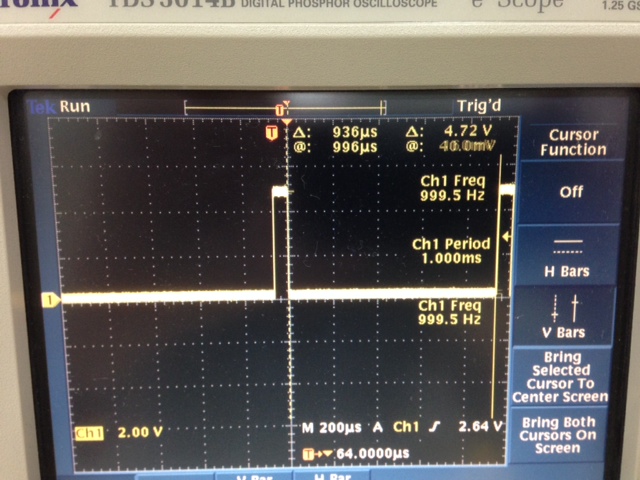


Figure 16

# Integration Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.11** |  |
| **Requirements covered** | | |
| 1.15 | | |
| **Test Procedure** | | |
| Turn a Pin level ON at the entrance of a task and turn the Pin level OFF at the end of a task execution | | |
| **Expected Results** | | |
| Using an oscilloscope verify that every Task is working at the specified period | | |
| **Actual Results** | | **Test Results** |
| See figure 17 | | PASS |
| **Comments** | | |
| As we could see in the next figure, there are different signals for each task, for the tick and for the background. We can see in the figure 17 how each task is activated at different time( 1ms, 4ms, 8 ms, 16 ms, 32 ms and 64ms) | | |

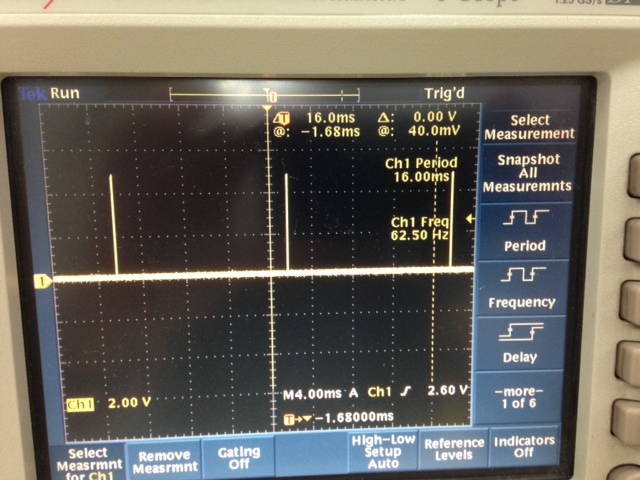
Next figures show each task period



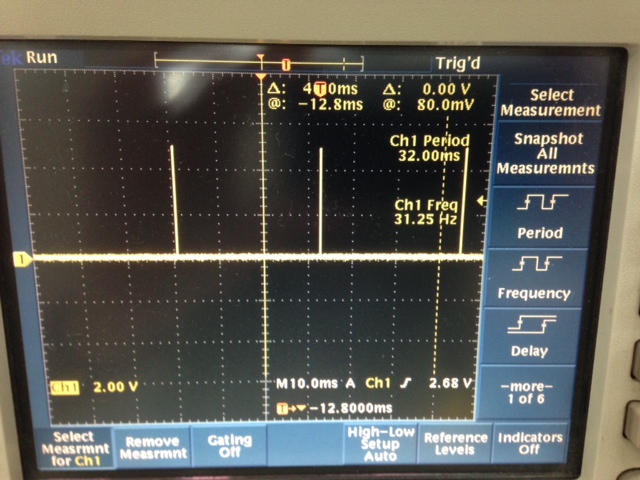
Task 1ms



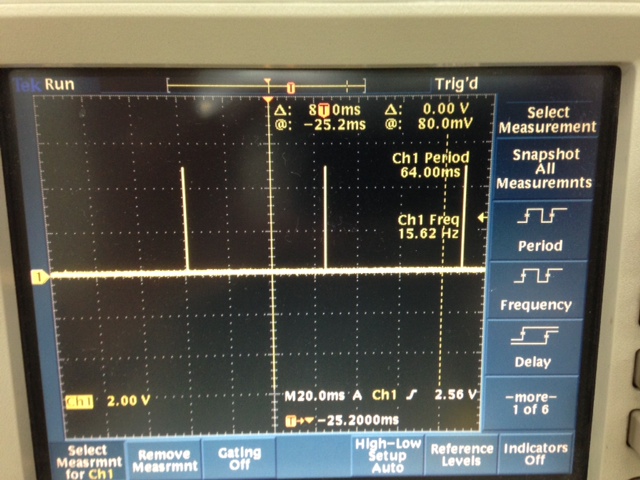
Task 4ms



Task 16ms



Task 32 ms



Task 64ms

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.12** |  |
| **Requirements covered** | | |
| 1.16 | | |
| **Test Procedure** | | |
| Turn a pin level ON before entering the Background Task and turn the pin level OFF at the end of the Background Task | | |
| **Expected Results** | | |
| Using an oscilloscope verify the CPU Load at the specified pin level | | |
| **Actual Results** | | **Test Results** |
| See figure 17 | | PASS |
| **Comments** | | |
| As we could see in the next figure, there are different signals for each task, for the tick and for the background. To consider that this test case passed, we could see that background signal is activated every time a task is executed. So, background signal is aligned with each width of each task. | | |



Figure 17

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
|  | **1.13** |  |
| **Requirements covered** | | |
| 1.17 | | |
| **Test Procedure** | | |
| Modify the CPU Load by adding workload to the tasks | | |
| **Expected Results** | | |
| Using an oscilloscope verify the CPU Load at the specified pin level | | |
| **Actual Results** | | **Test Results** |
| See figure 18 and figure 19 | | PASS |
| **Comments** | | |
| As explained in the Test case ID 1.12, each task has a row in the next figures 18 and 19. For these images we modified the workload of each task, you could see the differences in each task for both images. It can be clearly seen in the width of each pulse in each task. Again, in the background task we see how the width of the pulse of the activated task in that time is aligned with the width of the background task. | | |

****

**Figure 18**

****

**Figure 19**