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

|  |  |
| --- | --- |
| Nr. : | 01 |
| Title: | Context\_Switch |

**Contents**

1. Test Specification Information 3

2. Module Test Cases 3

3. Integration Test Cases 12

# Test Specification Information

|  |  |  |
| --- | --- | --- |
| **Date of issue (MM/DD/YY)** | **Test Developer** | **Revision & Description** |
| 25/03/2014 | Miguel, Sergio, Esteban | 2.0 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Module Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| File Structure | **2.0** | **Done** |
| **Requirements covered** | | |
| 3.0 | | |
| **Test Procedure** | | |
| The void DisableAllInterrupts ( void ) API service saves the current state of all interrupts, disables all interrupts that are enabled, and identifies the beginning of a critical section.  - Within the critical section, no API service calls are allowed.  - How the system interrupts are disabled will differ between  implementations and between microcontrollers | | |
| **Expected Results** | | |
| Interrupts Disabled | | |
| **Actual Results** | | **Test Results** |
| Interrupts Disabled during critical code execution. See Figure 1, Figure 2 | | PASS |
| **Comments** | | |
|  | | |

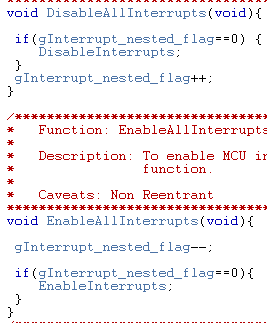


Figure 1



Figure 2

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Task Manager Services | **2.1** | **Done** |
| **Requirements covered** | | |
| 3.1 | | |
| **Test Procedure** | | |
| The void EnableAllInterrupts ( void ) API service enables all interrupts that were enabled prior to the previous call to DisableAllInterrupts() and identifies the end of a critical section.  - Within the critical section, no API service calls are allowed.  - If DisableAllInterrupts() was not previously called, the action taken is undefined by the specification. | | |
| **Expected Results** | | |
| Interrupts Enabled | | |
| **Actual Results** | | **Test Results** |
| Interrupts Enabled See Figure 2 & Figure 3 | | PASS |
| **Comments** | | |
|  | | |

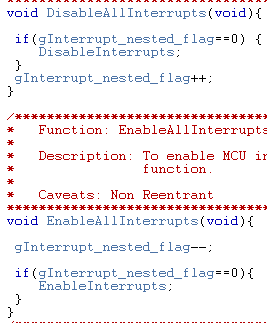


Figure 2

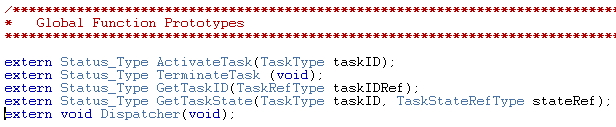


Figure 3

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Configuration Support | **2.2** | **Done** |
| **Requirements covered** | | |
| 3.3 | | |
| **Test Procedure** | | |
| DisableAllInterrupts and EnableAllInterrupts system services shall support nested calls in order to avoid any enabling interrupt when not yet required. | | |
| **Expected Results** | | |
| Interrupts Disabled/Enabled Nested | | |
| **Actual Results** | | **Test Results** |
| Interrupts Disabled/Enabled Nested | | PASS |
| **Comments** | | |
| See Figure 4 | | |

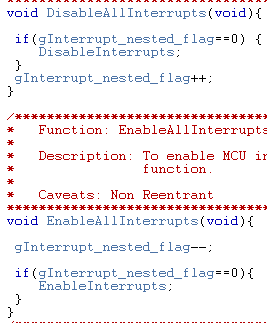


Figure 4

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Task Configuration | **2.3** | **Done** |
| **Requirements covered** | | |
| 3.9 | | |
| **Test Procedure** | | |
| Dispatcher shall only be called at the end of the following  system services:  – ActivateTask  – TerminateTask  – End of CAT 2 ISR’s | | |
| **Expected Results** | | |
| Dispatcher Called Only after Activate Task, Terminate Task and ISR. | | |
| **Actual Results** | | **Test Results** |
| Dispatcher Called Only after Activate Task, Terminate Task and ISR. | | PASS |
| **Comments** | | |
| See Figure 5 & Figure 6 | | |

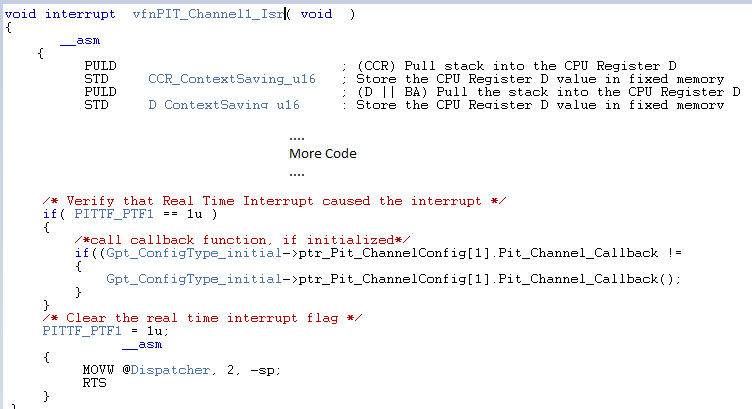


Figure 5

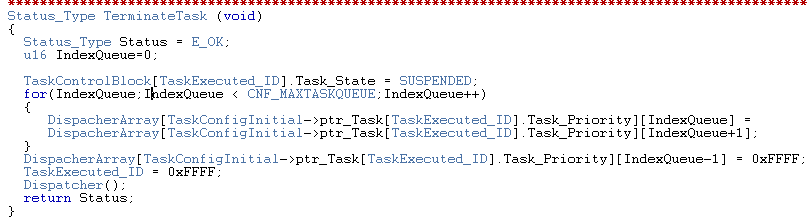


Figure 6

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Data definitions | **2.4** | **Done** |
| **Requirements covered** | | |
| 3.10, 3.11,3.12,3.13,3.14,3.22 | | |
| **Test Procedure** | | |
| If higher priority task than the current running task is ready for  execution, the dispatcher shall switch tasks context and run the  new task  - Previous task shall be queued in its corresponding priority  buffer  - Task states shall be changed accordingly | | |
| **Expected Results** | | |
| Task must be preempt accordantly their priority. | | |
| **Actual Results** | | **Test Results** |
| Task are preempt accordantly their priority. | | PASS |
| **Comments** | | |
| See Figure 7, Figure 8, Figure 9 | | |



Figure 7

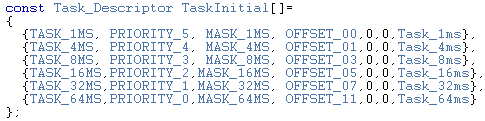


Figure 8

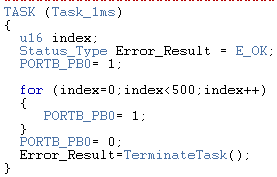
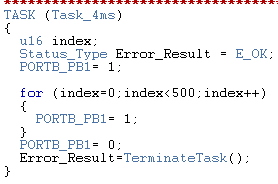
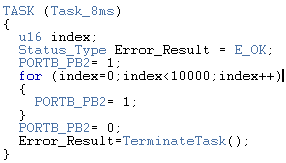
  

Figure 9

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Additional Information | **2.5** | **Done** |
| **Requirements covered** | | |
| 3.15,3.16,3.17 | | |
| **Test Procedure** | | |
| Task Stack shall be allocated with Memory Allocation interface | | |
| **Expected Results** | | |
| Task Stack shall be allocated with Memory Allocation interface | | |
| **Actual Results** | | **Test Results** |
| Task Stack is be allocated with Memory Allocation interface | | PASS |
| **Comments** | | |
| See Figure 10 | | |

SchM.c

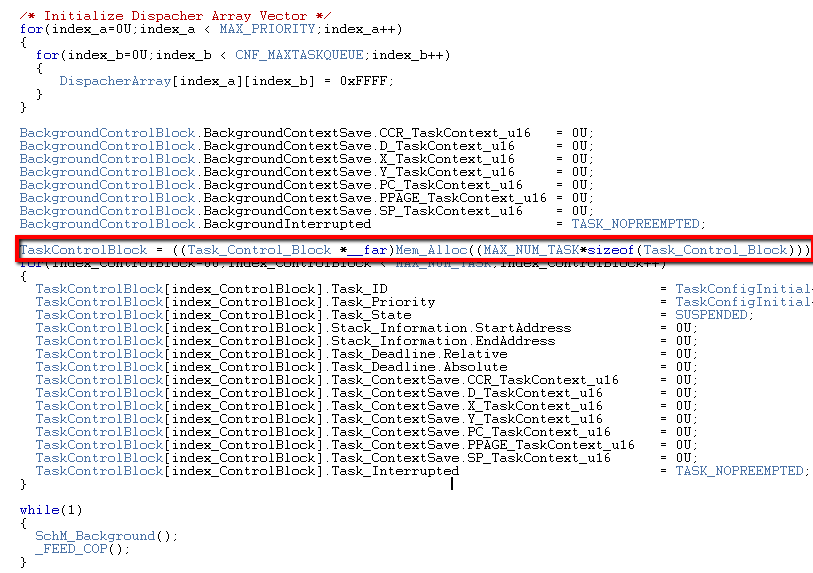


Figure 10

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Scheduler Services | **2.6** | **Done** |
| **Requirements covered** | | |
| 3.21 | | |
| **Test Procedure** | | |
| Background Task shall execute RotaBit algorithm using LED 0 ..  LED 3 (PA0 .. PA3) from EP100 Board.  – Increment a counter on the event driven task Timed Task 1  – This counter shall served as timer counter for the RotaBit algorithm | | |
| **Expected Results** | | |
| Background Task shall execute RotaBit algorithm using LED 0 ..  LED 3 (PA0 .. PA3) from EP100 Board.  – Increment a counter on the event driven task Timed Task 1  – This counter shall served as timer counter for the RotaBit algorithm | | |
| **Actual Results** | | **Test Results** |
| Background Task is execute RotaBit algorithm using LED 0 ..  LED 3 (PA0 .. PA3) from EP100 Board.  – Increment a counter on the event driven task Timed Task 1  – This counter shall served as timer counter for the RotaBit algorithm | | PASS |
| **Comments** | | |
| See Figure 11, Figure 12 | | |

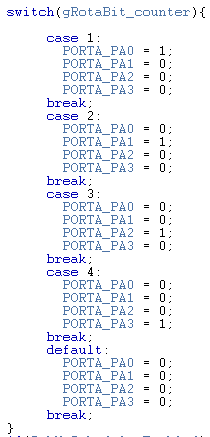


Figure 11

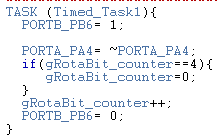


Figure 12

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Background | **2.7** | **Done** |
| **Requirements covered** | | |
| 2.13 | | |
| **Test Procedure** | | |
| Call SchM\_Start inside of main, then call SchM\_Background. | | |
| **Expected Results** | | |
| SchM\_Start must be called from main, after that SchM\_Background should be called and must never ends | | |
| **Actual Results** | | **Test Results** |
| SchM\_Background is called after SchM\_Start inside of main and never ends | | PASS |
| **Comments** | | |
| See Figure 13 & Figure 14 | | |

Main.c

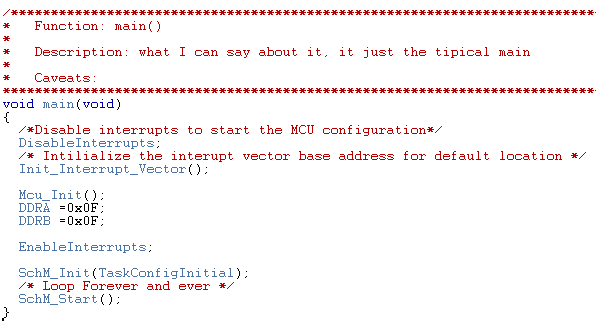


Figure 13

SchM.c

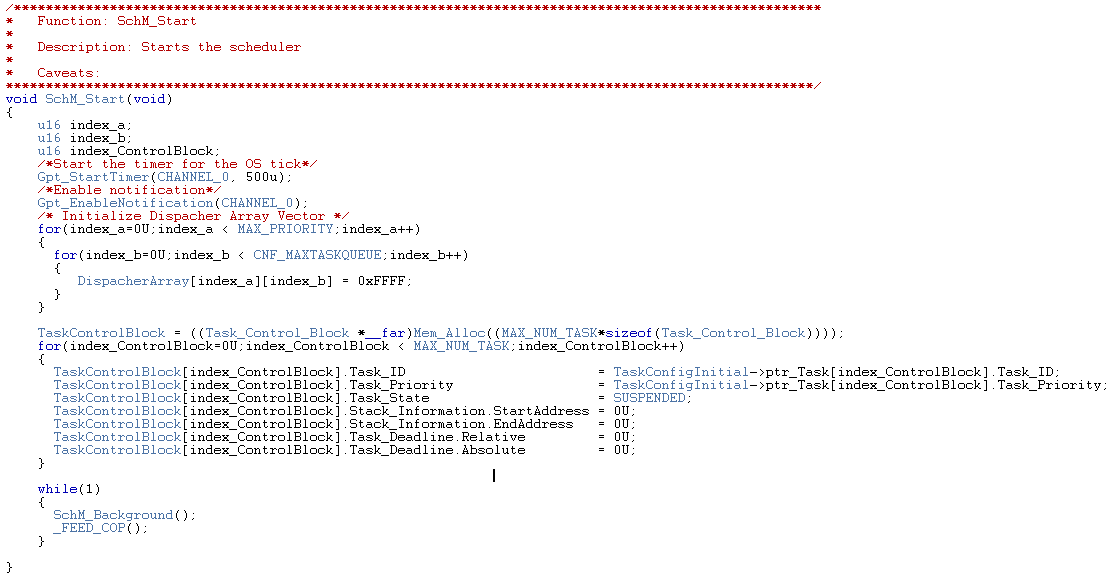


Figure 14

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Priority | **2.8** | **Done** |
| **Requirements covered** | | |
| 2.14, 2.15 | | |
| **Test Procedure** | | |
| Assign the lowest priority value to the lowest priority task, assign the same priority to a few tasks | | |
| **Expected Results** | | |
| Task execution can be performed according its priority assigned even if they have the same priority | | |
| **Actual Results** | | **Test Results** |
| Tasks are executed according its priority no matter whether or not they have the same priority | | PASS |
| **Comments** | | |
| Task priorities are configured according requirement 2.14 initially, but tasks sharing the same priority according requirement 2.15, are fully supported. See Figure 15 | | |

Os\_TaskCfg.c

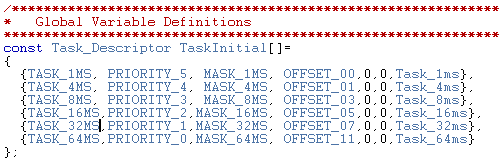
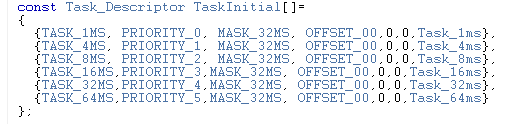


Figure 15

# Integration Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Running project | **2.9** | **Done** |
| **Requirements covered** | | |
| 2.16, 2.17 | | |
| **Test Procedure** | | |
| Set a pin level to high when tasks start its execution. Set a pin level low when tasks end its execution. | | |
| **Expected Results** | | |
| Pin out level must be set high every time tasks start its execution, and must be set low every time the tasks ends its execution. | | |
| **Actual Results** | | **Test Results** |
| See the next figures which contains different configurations for the task properties. | | PASS |
| **Comments** | | |
| Each result is composed with two images, the first shows the configuration of the task and the second shows the result and order in which the task were executed. | | |

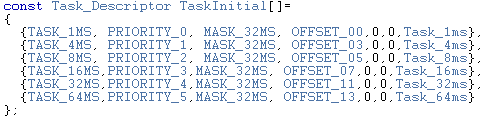
Next figures show each result.



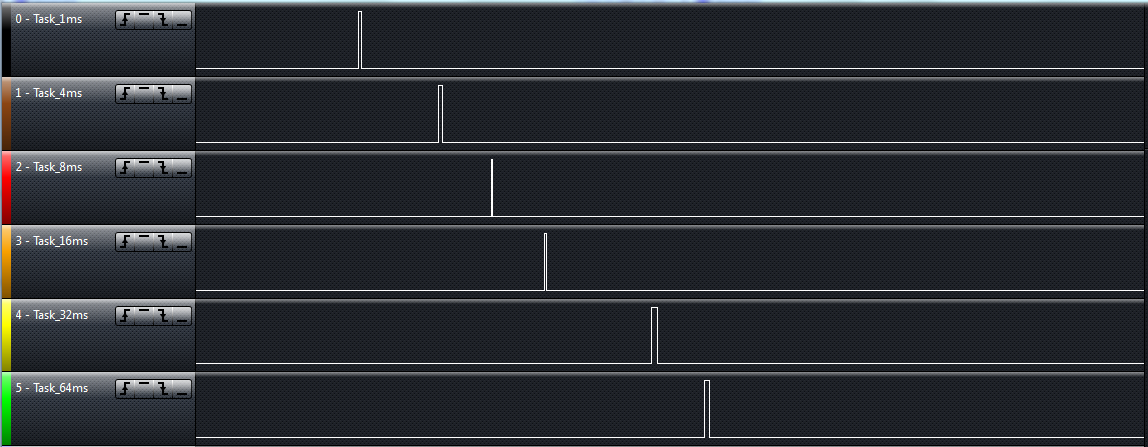
Code Test 1



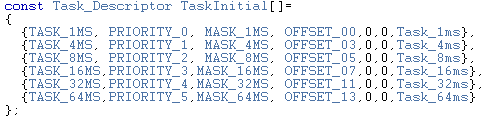
Result Test 1



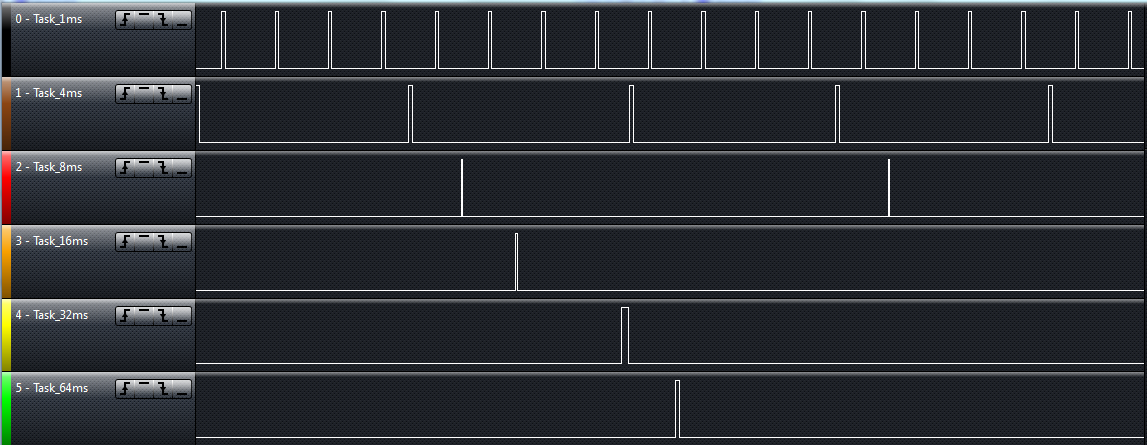
Code Test 2



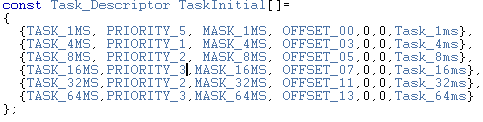
Result Test 2



Code Test 3



Result Test 3



Code Test 4



Result Test 4

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Running project | **2.10** | **Done** |
| **Requirements covered** | | |
| 2.18 | | |
| **Test Procedure** | | |
| Define the basic states Suspended, Ready & Running for all Tasks States. | | |
| **Expected Results** | | |
| Tasks states will be transitioning the basic states Suspended, Ready & Running. | | |
| **Actual Results** | | **Test Results** |
| See Figure 16 & Figure 17. | | PASS |
| **Comments** | | |
| Tasks will be set in suspended state every time they’re called to terminate in TerminateTask API.  Tasks in ready state will be set every time they’re put into the priority buffer by ActivateTask API.  Tasks in running state are set every time they’re called inside the Dispatcher API. | | |

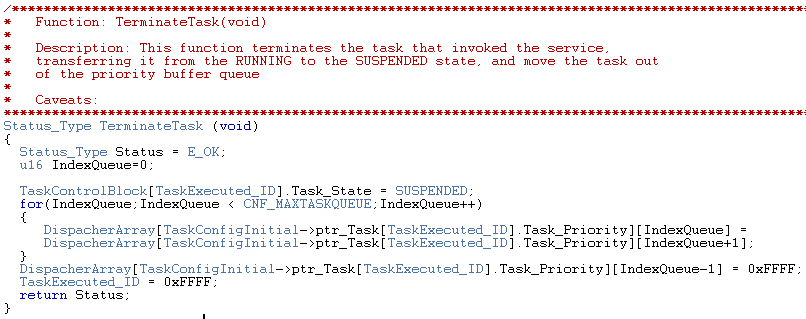
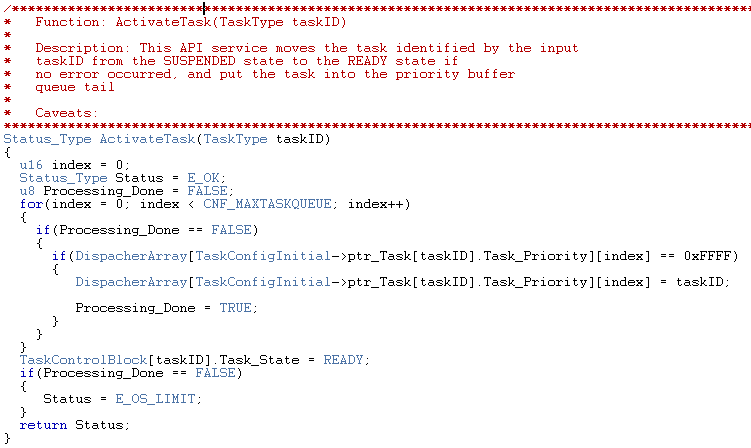


Figure 16



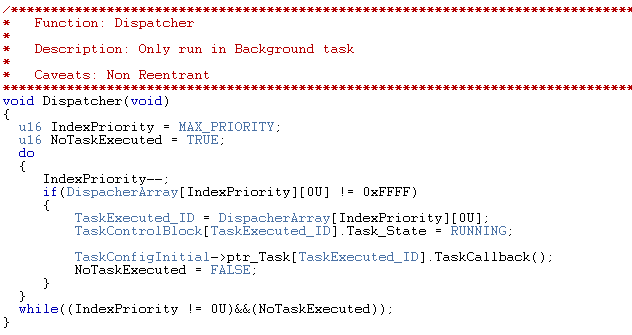


Figure 17

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Running project | **2.11** | **Done** |
| **Requirements covered** | | |
| 2.19 | | |
| **Test Procedure** | | |
| Place ready tasks IDs into its correspondent priority buffer. | | |
| **Expected Results** | | |
| Tasks IDs are placed into its correspondent priority buffer every time they turn ready. | | |
| **Actual Results** | | **Test Results** |
| Tasks IDs are placed into its correspondent priority buffer every time they turn ready. As figure 18 and 19. | | PASS |
| **Comments** | | |
|  | | |

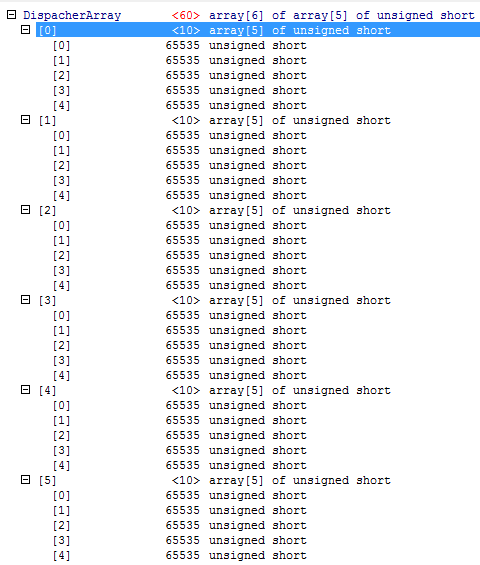


Figure 18

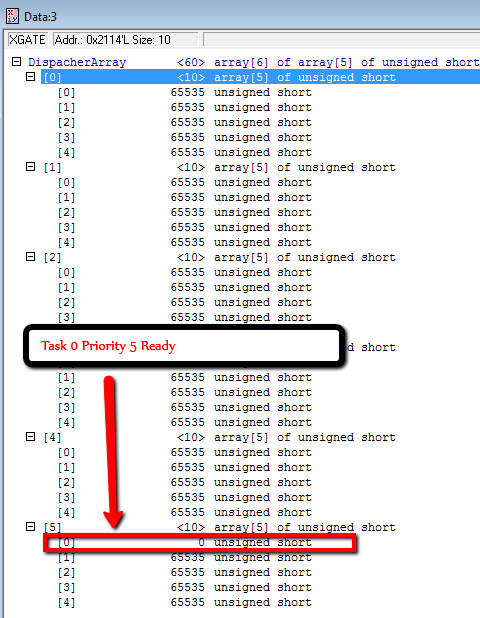


Figure 19

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Running project | **2.12** | **Done** |
| **Requirements covered** | | |
| 2.20 | | |
| **Test Procedure** | | |
| Move out tasks which transition from ready to suspended from its correspondent priority buffer | | |
| **Expected Results** | | |
| Tasks to be suspended are moved out from its last priority buffer. | | |
| **Actual Results** | | **Test Results** |
| Tasks to be suspended are moved out from its last priority buffer. As figure 20 and 21 | | PASS |
| **Comments** | | |
|  | | |

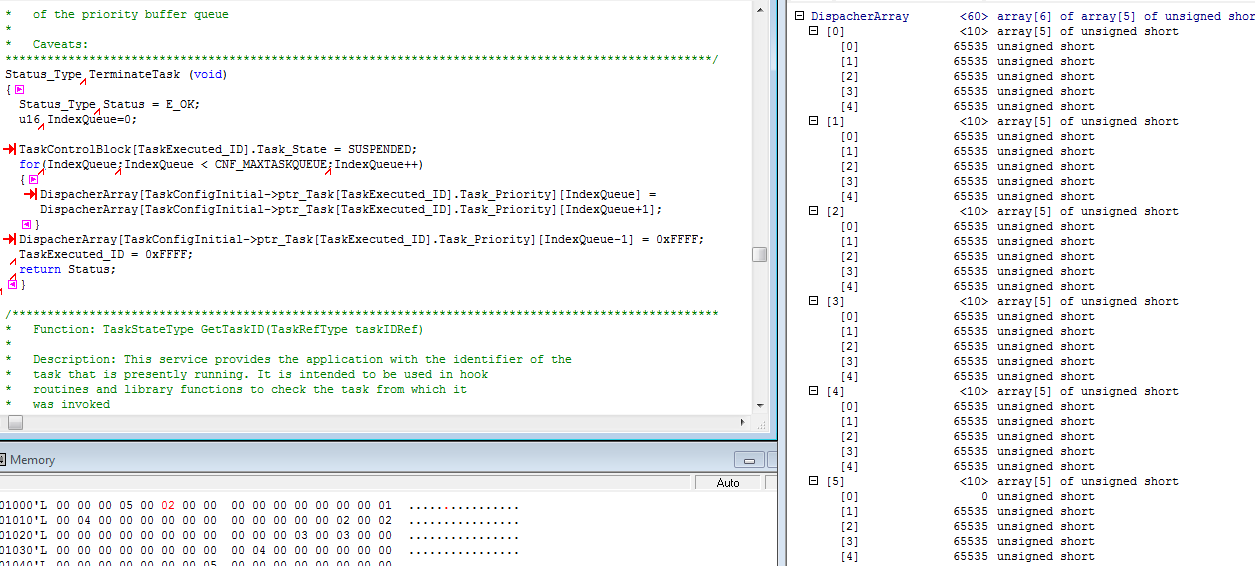


Figure 20

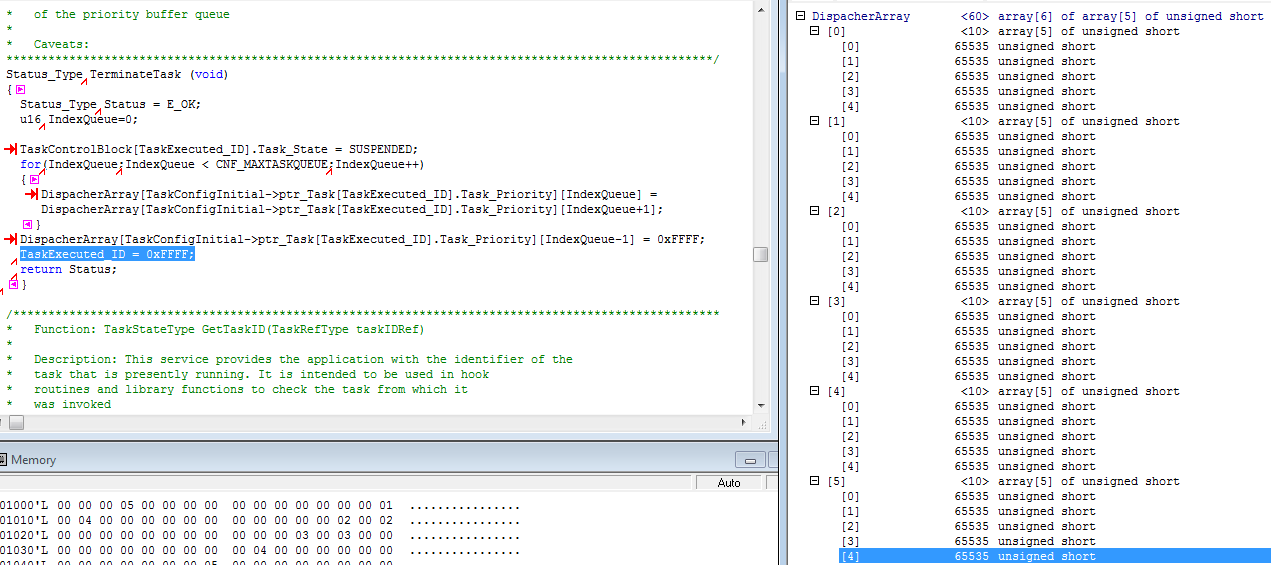


Figure 21

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Internal requirements | **2.13** | **Done** |
| **Requirements covered** | | |
| 2.21 | | |
| **Test Procedure** | | |
| Include memory allocation driver from previous projects. | | |
| **Expected Results** | | |
| Memory allocation will be available for usage in Task manager project. | | |
| **Actual Results** | | **Test Results** |
| Memory allocation will be available for usage in Task manager project. As figure 22 and 23 | | PASS |
| **Comments** | | |
|  | | |

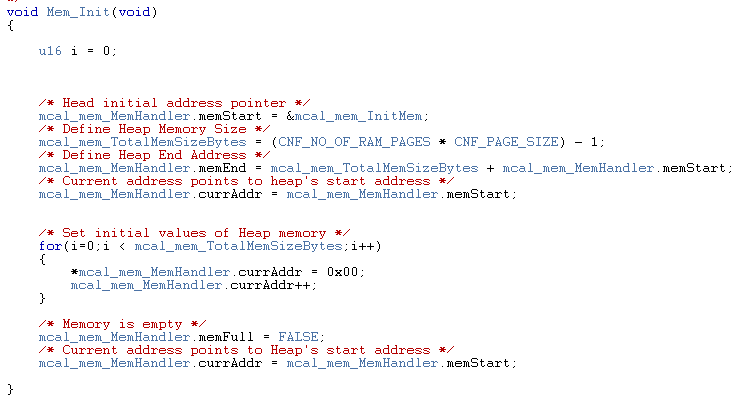


Figure 22



Figure 23

|  |  |  |
| --- | --- | --- |
| **Test Case** | **ID** | **Status** |
| Internal requirements | **2.14** | **Done** |
| **Requirements covered** | | |
| 2.22 | | |
| **Test Procedure** | | |
| Reserve memory for control block structure using memory allocation. | | |
| **Expected Results** | | |
| Reserved memory section is initialized to 0 and its size is the control block structure size. | | |
| **Actual Results** | | **Test Results** |
| Reserved memory section is initialized to 0 and its size is the control block structure size. As figure 24 and 25 | | PASS |
| **Comments** | | |
|  | | |

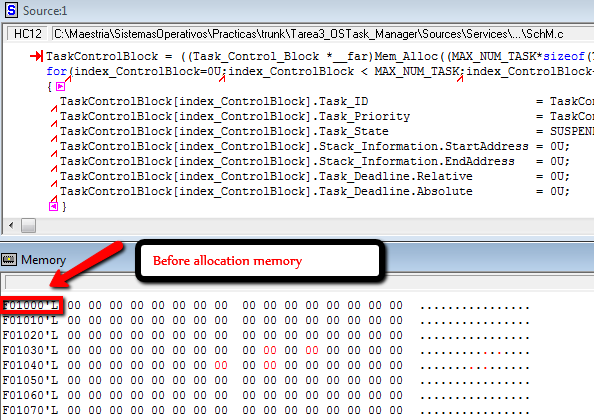


Figure 24

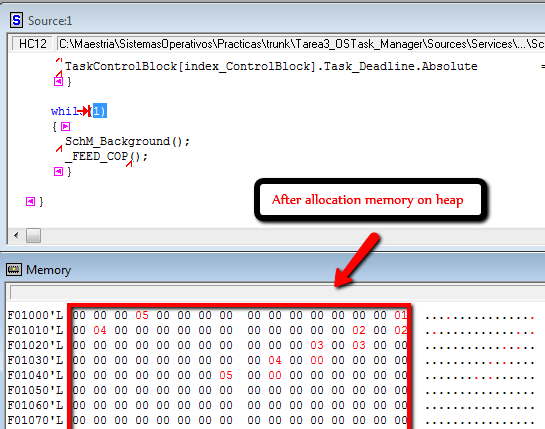


Figure 25