RENESAS Tool News

RENESAS TOOL NEWS on July 1, 2007: 070701/tn6

A Note on Using the Real-Time OS--M3T-MR30/4 V.4.00 Release 00-- for the M16C MCU Series

Please take note of the following problem in using the real-time OS-- M3T-MR30/4 V.4.00 Release 00--for the M16C MCU series:*
*M16C is the generic name of the M16C/60, /30, /20, /10, /Tiny series.

- With using the ter_tsk service call

1. Description

If the ter_tsk service call is issued to a task that has been entered in the waiting state by a service call with a timeout value,* the task is not removed from the queue for objects, which causes such a symptom that is shown in Example 1 or 2 in Section 2 to arise.

*twai_sem, twai_flg, trcv_mbx, tsnd_dtq, trcv_dtq, tget_mpf, vtsnd_dtq, and vtrcv_dtq

2. Symptoms

Example 1

```
-------------------------------------------
 void main( VP_INT stacd )
 {
   sta_tsk(ID_task1,0);
   sta_tsk(ID_task2,0);

   ter_tsk(ID_task1);
   sig_sem(ID_sem1);          /* (1) */
 }
 void task1( VP_INT stacd )
 {
    twai_sem(ID_sem1,100);
 }
```

```
void task2( VP_INT stacd )
{
   wai_sem(ID_sem1);
}
```

If the sig_sem service call is issued (as indicated by comment (1) above), normally task2 is released from the WAINTING state for a semaphore and enters the RUNNING state. However, because the symptom forced task2 to keep the WAITING state for a semaphore, task1 enters the RUNNING state instead. Note that task1 has the highest priority among the tasks; task2 and main follows in this order.

Example 2

```
void main( VP_INT stacd )
{
  sta_tsk(ID_task1,0);
  sta_tsk(ID_task2,0);

  ter_tsk(ID_task1);
  sta_tsk(ID_task1,0);        /* (1) */
  while(1){}                  /* (2) */
}
void task1( VP_INT stacd )
{
   twai_sem(ID_sem1,100);
}

void task2( VP_INT stacd )
{
   wai_sem(ID_sem1);
}
void func(void)
{
   while(1){}                 /* (3) */
}
```

If the sta_tsk service call is issued (as indicated by comment (1) above), normally task1 and task2 enter the waiting state for a semaphore to execute while(1) in main (as indicated by comment (2) above). However, because the symptom destroys the semaphore-waiting and Ready queues, task2 continues to be executed though it is in the waiting state.

As a result, while(1) in func, which follows task2 (as indicated by comment (3) above) and must not be executed normally, will be executed to cause the system to run away.
Note that here also task1 has the highest priority among the tasks; task2 and main follows in this order.

3. Conditions

This problem occurs if the following conditions are all satisfied:
(1) The ter_tsk service call is used.
(2) The task to which the ter_tsk service call is issued has been entered in the waiting state by any of the following service calls:
twai_sem, twai_flg, trcv_mbx, tsnd_dtq, trcv_dtq, tget_mpf, vtsnd_dtq, and vtrcv_dtq

4. Workaround

Perform the following steps to avoid this problem:
(1) Issue the dis_dsp service call to disable dispatching.
This prevents the rel_wai service call from task switching.
(2) Issue the rel_wai service call to release the task involved from the waiting state forcibly.
(3) Issue the ter_tsk service call to the task involved.
(4) Issue the ena_dsp service call to enable dispatching.

Example

```
-----------------------------------------
 void main( VP_INT stacd )
 {
   sta_tsk(ID_task1,0);

   dis_dsp();              /* Step (1) */
   rel_wai(ID_task1);      /* Step (2) */
   ter_tsk(ID_task1);      /* Step (3) */
   ena_dsp();              /* Step (4) */

 }
 void task1( VP_INT stacd )
 {
    twai_sem(ID_sem1,100);
 }
-----------------------------------------
```

5. Schedule of Fixing the Problem

We plan to fix this problem in the next release of the product.
