Freescale MQX RTOS Example Guide

access_usr

This document describes the access_usr which demonstrates use of user-mode tasks and memory protection on the Kinetis platforms.

Pre-requisities

This example requires the support for User Mode to be compiled in the MQX kernel. Edit the <mqx_installation>/config/<board>/user_config.h file and add the

```
#define MQX_ENABLE_USER_MODE 1
```

Then rebuild the MQX as described in the MQX Getting Started document. **Also make sure the board jumpers are set properly as described in this document.**

Note that this application uses a different linker command file than the other examples which do not make use of User mode. The linker file for User mode applications defines additional memory areas for different levels of protection as well as the area for the memory heap used by User mode tasks. The linker files suitable for the User mode examples are named with the _usr postfix.

Note that not all build tools and processor platforms are supported by the User-mode feature.

More Reading

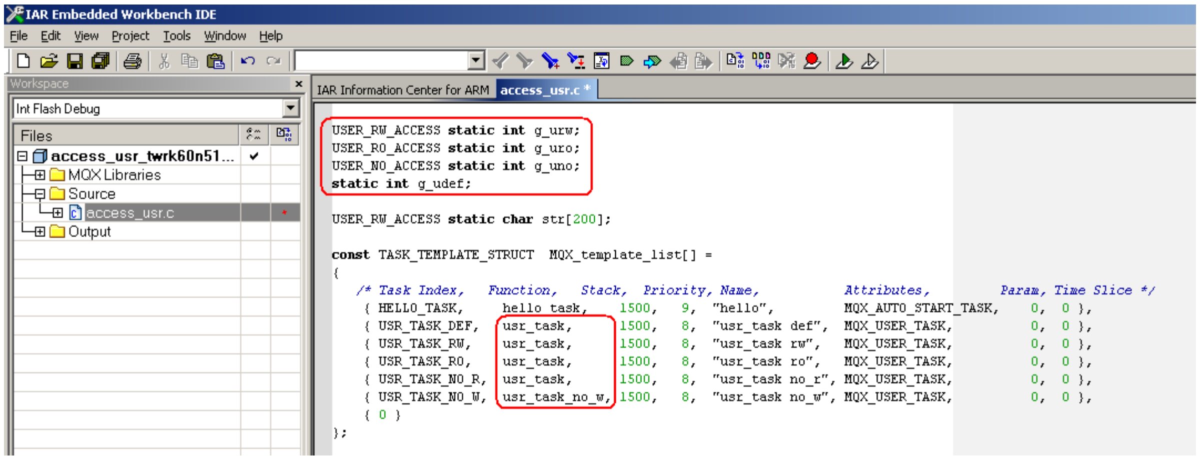
Read more information about the User vs. Privileged execution mode and about memory protection in the MQX RTOS User Guide. See also User mode API in the MQX API Reference Manual. The User mode functions all have the *_usr* prefix.

The Example

The example exercises the memory access in different areas from a User mode task. See the definition of global variables, each defined in different access area:

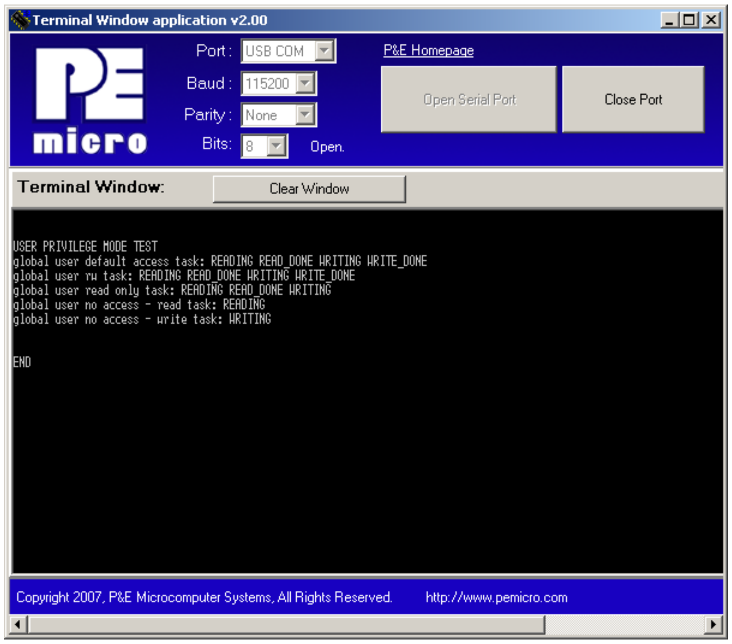
- **g_urw** – full read/write access from User mode task
- **g_uro** – read-only access from User mode task
- **g_uno** – no access from User mode task
- **g_udef** – access is not explicitly defined, it is either read/write or read-only, depending on the MQX_DEFAULT_USER_ACCESS_RW kernel configuration option

One instance of a user-task is started for each variable. The task simply tries to read the variable, increment and write the value back. Depending on the memory protection a task instance either passes through both variable accesses or gets terminated upon a memory violation exception.
The task progress is recorded in a string variable using a sprintf method. The string variable is later printed to the console by the main task.



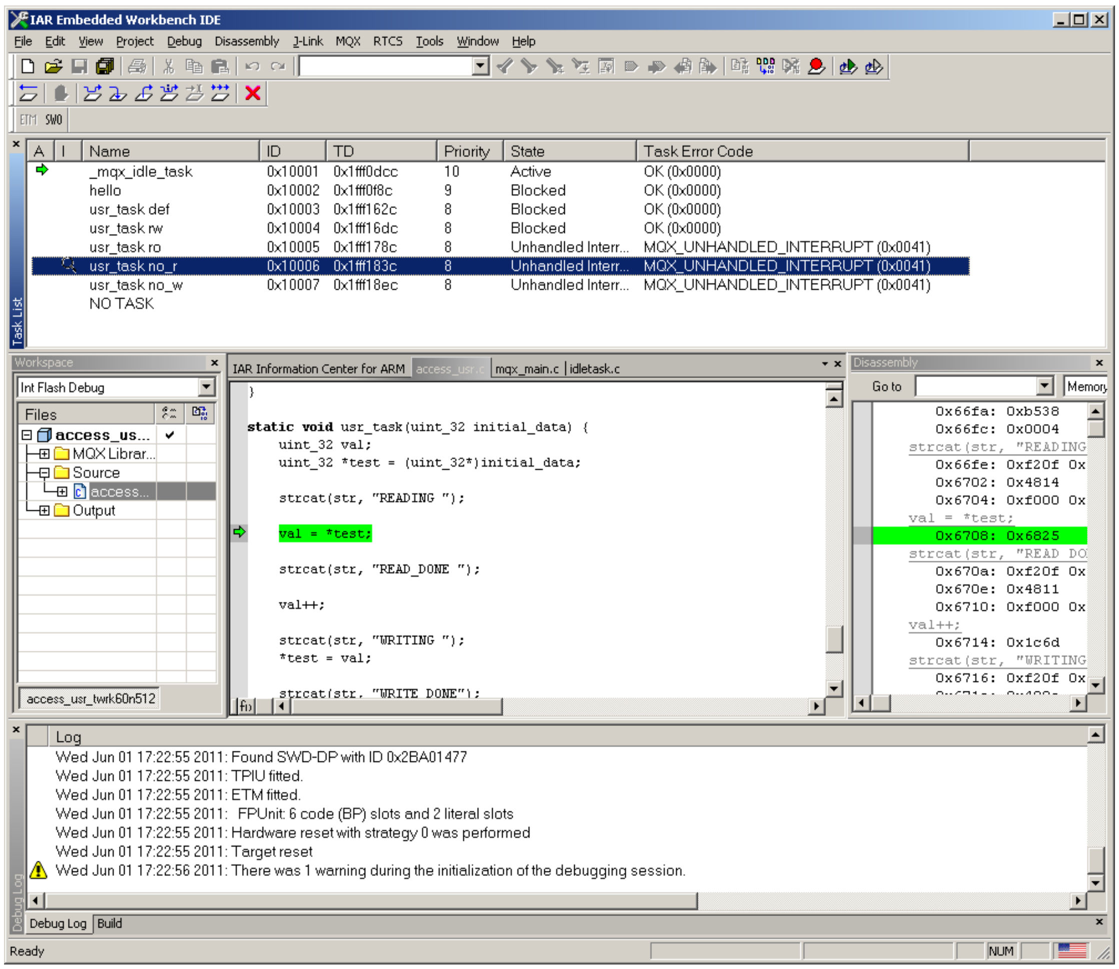
```
USER_RW_ACCESS static int g_urw;
USER_RO_ACCESS static int g_uro;
USER_NO_ACCESS static int g_uno;
static int g_udef;

USER_RW_ACCESS static char str[200];

const TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
   /* Task Index,   Function,   Stack,  Priority, Name,            Attributes,          Param, Time Slice */
    { HELLO_TASK,     hello_task,    1500,    9,  "hello",         MQX_AUTO_START_TASK,    0,   0 },
    { USR_TASK_DEF,  usr_task,       1500,    8,  "usr_task def",  MQX_USER_TASK,          0,   0 },
    { USR_TASK_RW,   usr_task,       1500,    8,  "usr_task rw",   MQX_USER_TASK,          0,   0 },
    { USR_TASK_RO,   usr_task,       1500,    8,  "usr_task ro",   MQX_USER_TASK,          0,   0 },
    { USR_TASK_NO_R, usr_task,       1500,    8,  "usr_task no_r", MQX_USER_TASK,          0,   0 },
    { USR_TASK_NO_W, usr_task_no_w,  1500,    8,  "usr_task no_w", MQX_USER_TASK,          0,   0 },
    { 0 }
};
```

With the User-Mode support enabled you should see the user tasks are terminated upon an illegal memory access. The console should display something like this:



```
USER PRIVILEGE MODE TEST
global user default access task: READING READ_DONE WRITING WRITE_DONE
global user rw task: READING READ_DONE WRITING WRITE_DONE
global user read only task: READING READ_DONE WRITING
global user no access - read task: READING
global user no access - write task: WRITING

END
```

The tasks pass through or are terminated properly upon illegal memory access. You can also examine the task status in the debugger session. The MQX Task List view should display tasks terminated at unhandled exception. If you double click the task item, the IAR debugger displays the code location where the task got blocked at unhandled exception.



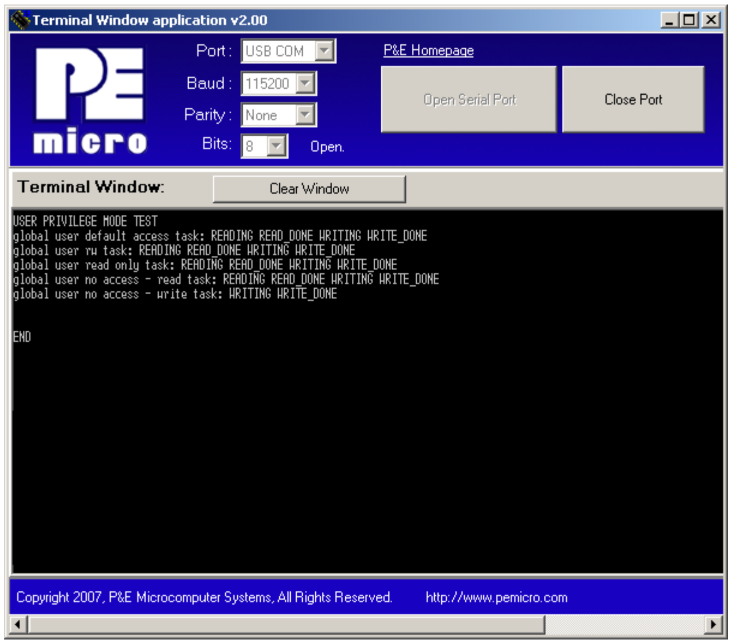
```
A  I  Name            ID       TD          Priority  State              Task Error Code
      _mqx_idle_task  0x10001  0x1fff0dcc  10        Active             OK (0x0000)
      hello           0x10002  0x1fff0f8c  9         Blocked            OK (0x0000)
      usr_task def    0x10003  0x1fff162c  8         Blocked            OK (0x0000)
      usr_task rw     0x10004  0x1fff16dc  8         Blocked            OK (0x0000)
      usr_task ro     0x10005  0x1fff178c  8         Unhandled Interr...  MQX_UNHANDLED_INTERRUPT (0x0041)
      usr_task no_r   0x10006  0x1fff183c  8         Unhandled Interr...  MQX_UNHANDLED_INTERRUPT (0x0041)
      usr_task no_w   0x10007  0x1fff18ec  8         Unhandled Interr...  MQX_UNHANDLED_INTERRUPT (0x0041)
      NO TASK
```

```
}

static void usr_task(uint_32 initial_data) {
    uint_32 val;
    uint_32 *test = (uint_32*)initial_data;

    strcat(str, "READING ");

    val = *test;

    strcat(str, "READ_DONE ");

    val++;

    strcat(str, "WRITING ");
    *test = val;

    strcat(str, "WRITE DONE");
```

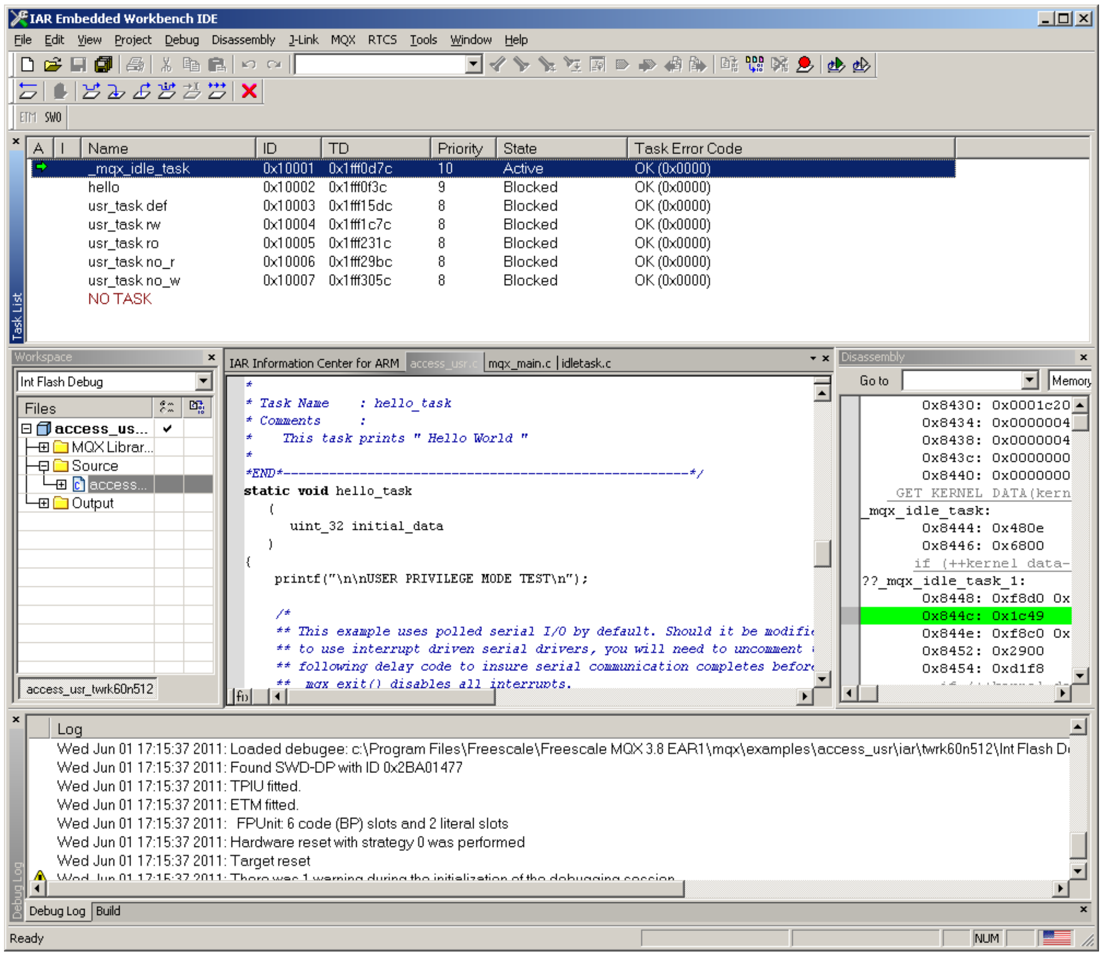
```
    0x66fa: 0xb538
    0x66fc: 0x0004
strcat(str, "READING
    0x66fe: 0xf20f 0x
    0x6702: 0x4814
    0x6704: 0xf000 0x
val = *test;
    0x6708: 0x6825
strcat(str, "READ DO
    0x670a: 0xf20f 0x
    0x670e: 0x4811
    0x6710: 0xf000 0x
val++;
    0x6714: 0x1c6d
strcat(str, "WRITING
    0x6716: 0xf20f 0x
```

```
Log
Wed Jun 01 17:22:55 2011: Found SWD-DP with ID 0x2BA01477
Wed Jun 01 17:22:55 2011: TPIU fitted.
Wed Jun 01 17:22:55 2011: ETM fitted.
Wed Jun 01 17:22:55 2011:   FPUnit: 6 code (BP) slots and 2 literal slots
Wed Jun 01 17:22:55 2011: Hardware reset with strategy 0 was performed
Wed Jun 01 17:22:55 2011: Target reset
Wed Jun 01 17:22:56 2011: There was 1 warning during the initialization of the debugging session.
```

Just for a completeness: Without the MQX_ENABLE_USER_MODE enabled in the MQX kernel, there is no memory protection active and all tasks will succeed in both memory operations:



```
USER PRIVILEGE MODE TEST
global user default access task: READING READ_DONE WRITING WRITE_DONE
global user rw task: READING READ_DONE WRITING WRITE_DONE
global user read only task: READING READ_DONE WRITING WRITE_DONE
global user no access - read task: READING READ_DONE WRITING WRITE_DONE
global user no access - write task: WRITING WRITE_DONE


END
```



A	I	Name	ID	TD	Priority	State	Task Error Code
		_mqx_idle_task	0x10001	0x1fff0d7c	10	Active	OK (0x0000)
		hello	0x10002	0x1fff0f3c	9	Blocked	OK (0x0000)
		usr_task def	0x10003	0x1fff15dc	8	Blocked	OK (0x0000)
		usr_task rw	0x10004	0x1fff1c7c	8	Blocked	OK (0x0000)
		usr_task ro	0x10005	0x1fff231c	8	Blocked	OK (0x0000)
		usr_task no_r	0x10006	0x1fff29bc	8	Blocked	OK (0x0000)
		usr_task no_w	0x10007	0x1fff305c	8	Blocked	OK (0x0000)
		NO TASK					

```
*
* Task Name    : hello_task
* Comments     :
*    This task prints " Hello World "
*
*END*----------------------------------------------------------*/
static void hello_task
   (
      uint_32 initial_data
   )
{
    printf("\n\nUSER PRIVILEGE MODE TEST\n");

    /*
    ** This example uses polled serial I/O by default. Should it be modifie
    ** to use interrupt driven serial drivers, you will need to uncomment 
    ** following delay code to insure serial communication completes befor
    **  mqx exit() disables all interrupts.
```

```
          0x8430: 0x0001c20
          0x8434: 0x0000004
          0x8438: 0x0000004
          0x843c: 0x0000000
          0x8440: 0x0000000
       GET KERNEL DATA(kern
_mqx_idle_task:
          0x8444: 0x480e
          0x8446: 0x6800
            if (++kernel data-
??_mqx_idle_task_1:
          0x8448: 0xf8d0 0x
          0x844c: 0x1c49
          0x844e: 0xf8c0 0x
          0x8452: 0x2900
          0x8454: 0xd1f8
```

```
Log
Wed Jun 01 17:15:37 2011: Loaded debugee: c:\Program Files\Freescale\Freescale MQX 3.8 EAR1\mqx\examples\access_usr\iar\twrk60n512\Int Flash D
Wed Jun 01 17:15:37 2011: Found SWD-DP with ID 0x2BA01477
Wed Jun 01 17:15:37 2011: TPIU fitted.
Wed Jun 01 17:15:37 2011: ETM fitted.
Wed Jun 01 17:15:37 2011:   FPUnit: 6 code (BP) slots and 2 literal slots
Wed Jun 01 17:15:37 2011: Hardware reset with strategy 0 was performed
Wed Jun 01 17:15:37 2011: Target reset
Wed Jun 01 17:15:37 2011: There was 1 warning during the initialization of the debugging session.
```