Freescale MQX Example Guide
core_mutex example

This document describes the core_mutex example application. The example allocates and creates a core mutex that is used for inter-core mutual exclusion to a shared data structure placed in the shared memory. This memory area is shared between individual cores of the multicore system like Freescale Vybrid MCU. The example shows how to work with the core_mutex driver and how to lock/unlock the core mutex to get an access to shared data.

The same code is executed on both cores, i.e. both cores are performing the same instructions. First, the application attempts to create the core mutex. The core mutex is created after the first attempt, it means the core that attempts to create the core mutex as the second just receives the pointer to the core mutex structure already created by the core that attempts to create the core mutex as a first. When calling the _core_mutex_create() function the core number, the mutex number and the task queueing policy is specified. Returned pointer to the core mutex is used each time the mutex lock/unlock function is called later.

Afterwards, shared data structure is cleared. The shared data structure is declared as follows:

```
typedef struct shared_data_struct {
    uint_32 count;
    uint_32 core_count[2];
} SHARED_DATA_STRUCT, * SHARED_DATA_STRUCT_PTR;
```

In fact, structure members are software counters. Each time a core locks the mutex it increments the "count" counter and the respective core counter "core_count[x]". Then the core mutex is released, i.e. _core_mutex_unlock() function is called.

The state of all counters can be monitored by the serial console. The core with the core number 1 prints the state of all counters every 1000 iterations.
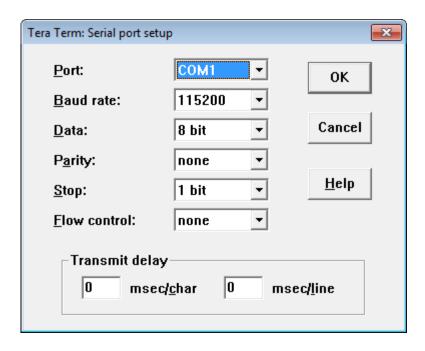
Running the example

Build the BSP and PSP projects for the target multicore platform. In case of Vybrid device build BSPs and PSPs for both A5 and M4 cores.

Refer to development tool – related MQX documentation for more information about multicore debugging (<MQX installation folder>\doc\tools\iar\MQX_IAR_Getting_Started.pdf).

Refer to "Freescale MQX I/O Drivers – User Guide" document for more information about the core_mutex driver (<MQX installation folder>\doc\mqx\MQX_IO_User_Guide.pdf).

Start a terminal application on your PC and set the serial connection for 115200 baud, 8 data bits, 1 stop bit, no parity and no flow control. Connect your PC with the target platform via the serial cable.

Tera Term: Serial port setup

Field	Value
Port:	COM1
Baud rate:	115200
Data:	8 bit
Parity:	none
Stop:	1 bit
Flow control:	none

OK Cancel Help

Transmit delay

0 msec/char 0 msec/line

Build and start core_mutex example applications on both cores of the target multicore platform. After starting both applications you will see the application log on the console.


```
COM1:115200baud - Tera Term VT
File  Edit  Setup  Control  Window  KanjiCode  Help
Count = 1104, core0_count=104, core1_count=1000
Count = 2214, core0_count=214, core1_count=2000
Count = 3317, core0_count=317, core1_count=3000
Count = 4422, core0_count=422, core1_count=4000
Count = 5526, core0_count=526, core1_count=5000
Count = 6631, core0_count=631, core1_count=6000
Count = 7737, core0_count=737, core1_count=7000
Count = 8843, core0_count=843, core1_count=8000
Count = 9942, core0_count=942, core1_count=9000
Count = 11050, core0_count=1050, core1_count=10000
Count = 12153, core0_count=1153, core1_count=11000
Count = 13260, core0_count=1260, core1_count=12000
Count = 14364, core0_count=1364, core1_count=13000
Count = 15474, core0_count=1474, core1_count=14000
Count = 16576, core0_count=1576, core1_count=15000
Count = 17679, core0_count=1679, core1_count=16000
Count = 18784, core0_count=1784, core1_count=17000
Count = 19894, core0_count=1894, core1_count=18000
Count = 20999, core0_count=1999, core1_count=19000
Count = 22105, core0_count=2105, core1_count=20000
Count = 23211, core0_count=2211, core1_count=21000
Count = 24319, core0_count=2319, core1_count=22000
Count = 25432, core0_count=2432, core1_count=23000
Count = 26537, core0_count=2537, core1_count=24000
Count = 27644, core0_count=2644, core1_count=25000
Count = 28751, core0_count=2751, core1_count=26000
Count = 29853, core0_count=2853, core1_count=27000
Count = 30959, core0_count=2959, core1_count=28000
Count = 32066, core0_count=3066, core1_count=29000
Count = 33173, core0_count=3173, core1_count=30000
Count = 34275, core0_count=3275, core1_count=31000
Count = 35378, core0_count=3378, core1_count=32000
```

test_task on core 0

```
test_task
on core 0
↓
Create core mutex
↓
Lock mutex
↓
Clear shared data structure
↓
Unlock mutex
↓
Lock mutex
↓
Increment "count" counter
↓
Increment "core_count[core num]" counter
↓
Unlock mutex
↓
corenum is 1 ──NO──┐
↓ YES              │
(iteration % 1000) == 0) ──NO──┤
↓ YES              │
Lock mutex         │
↓                  │
Print data         │
↓                  │
Unlock mutex ──────┘
```

test_task on core 1

```
test_task
on core 1
↓
Create core mutex
↓
Lock mutex
↓
Clear shared data structure
↓
Unlock mutex
↓
Lock mutex
↓
Increment "count" counter
↓
Increment "core_count[core num]" counter
↓
Unlock mutex
↓
corenum is 1 ──NO──┐
↓ YES              │
(iteration % 1000) == 0) ──NO──┤
↓ YES              │
Lock mutex         │
↓                  │
Print data         │
↓                  │
Unlock mutex ──────┘
```