Freescale MQX RTOS Example Guide

QSPI example

This document explains the QSPI example, what to expect from the example and a brief introduction to the API used.

The example

The example is applied only to Vybrid tower board and Vybrid autoevb tower board because these two boards have the quad SPI serial flash memory modules. The example shows the basic operations which can be applied to quad SPI serial flash memory via the quad SPI interface of the MCU. Specifically it examines the erase flash memory, the read large number of data bytes as well as the read one data byte from flash memory at a time and the write a long array of data bytes to flash memory operations.

Running the example

As mentioned above the Vybrid tower board and/or Vybrid autoevb tower board are applicable to this example. The BSPCFG_ENABLE_QUADSPI0 macro must be set to non-zero and the BSPCFG_ENABLE_FLASHX_QUADSPI0 macro must be set to zero in the user_config.h file prior to compilation of MQX libraries and the example itself.
To run the example the corresponding IDE, compiler, debugger and a terminal program are needed.

Explaining the example

This example consists of three source files. They are main.c, qspi_memory.c and qspi_memory.h. The qspi_memory.c and qspi_memory.h files contain the definition for constants and functions being used by tasks and functions in main.c. Again the functions called by parent functions in qspi_memory.c file are from the quad spi driver of MQX RTOS.
The example defines only one task in file main.c called main_task. The main_task is responsible for different jobs as described below.

- Open the connection to the quad SPI serial flash memory which allows the application to handle the flash memory using pointer.
- Collect the attributes of the serial flash memory for later use. The interested attributes are the flash base address in the memory map of the MCU, the flag indicating whether or not the serial flash supports parallel operation, the total size of the flash memory and the size of a sector inside the flash memory.
- Perform some example processes with serial flash memory modules via the quad SPI interface.
    - Erasing the whole serial flash memory using *memory_chip_erase()* function. The performance in time is recorded and displayed on the terminal.

```
-------------- QSPI driver example --------------
This example application demonstrates usage of QSPI driver.

******************************************************************************
Erase the first flash chip, for S25FL128S/256S, it might take 30s/60s....
******************************************************************************
QuadSPI Successfully Erase Flash

Erase whole flash 64 sec, 8 millisec
Finish erase all flash
```

- Reading 20 bytes of data from serial flash memory concurrently using function *memory_read_data()*. The output is shown on the terminal.

```
****************************************
*** Function Test <memory_read_data> ****
****************************************
From Flash 20008001: first 20 btyes
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xf
f 0xff
```

- Reading 20 bytes of data from serial flash memory consecutively using function *memory_ip_read_byte()* with a for loop iterating 20 times. The output is displayed on the terminal.
  We should see that the output data in two cases are similar.

```
****************************************
*** Function Test <memory_ip_read_byte> ****
****************************************
From Flash 20008001: first 20 bytes
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xf
f 0xff
```

- Writing 16 kilo bytes of data to serial flash memory using function *memory_write_data()*, the written data is the sequence 0x00, 0x01, …, 0xff. The statistic of writing procedure is recorded and shown on the terminal.

```
****************************************
*** Function Test <memory_write_data> ***
****************************************

data = 16384, Time spends on Flash write is 0 sec, 74 millisec, rate = 221 kbps
```

- The buffer used to keep data read from serial flash memory is reset before function *memory_read_data()* is called again to read out the data written previously to the serial flash memory. The statistic of reading process and the first 20 data bytes are displayed on the terminal.

```
data = 16384, Time spends on Flash read is 0 sec, 1 millisec, rate = 16384 kbps

memory_read_data read data from 20008001: first 20 bytes
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xf
f 0xff
```

- The first 20 data bytes written earlier are also read back using function *memory_ip_read_byte()* and a for loop iterating over 20 times. The output data is presented on the terminal.

```
****************************************
**** Compare Test <memory_ip_read_byte> ****
****************************************
memory_ip_read_byte from 20008001: first 20 bytes
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x1
2 0x13
```

We should note that the data bytes read from serial flash using *memory_read_data()*and *memory_ip_read_byte()* are different. This is because the CPU uses two different bus types IP and AHB to access the quad SPI interface module and there are two different reading methods associated with these two bus types. Please refer to the reference manual of the MCU for detail. However the IP access method always returns the correct data from serial flash.

- The whole sector containing a test address in serial flash memory is erased by calling function *memory_sector_erase()* and the erase is verified with following output.

```
****************************************
***** Function Test <memory_sector_erase> ******
****************************************
Sector erase 0x20008001. Sector [0x20000000 - 0x2000ffff] erase is successful!
```

- The *rw_compare_test()* function written in main.c file is invoked. Data is written into serial flash memory using write function and read back using read functions discussed before. The sequence of write data, read data and verification of read and write process are repeated for a number of different pairs of address and length of data sequence. The output for two pairs of different address with corresponding length of data sequence is shown in the next page as an example.

- The example finishes with the message.

```
All tests are passed!

-------------- End of example --------------
```

```
*****************************************
**** Compare Test <memory_read_data> ****
*****************************************
Write 0xAB to 0x20002000, len: 0x3
Parallel mode is enabled, prepare the data
Write 0xCD to 0x21002000, len: 0x3
Read from 0x20004000, len: 0x3


*****************************************
**** Compare Test <memory_read_byte> ****
*****************************************


*****************************************
**** Compare Test <memory_read_data> ****
*****************************************
crossing sectors!
Write 0xAB to 0x2000F000, len: 0x1000
Parallel mode is enabled, prepare the data
Write 0xCD to 0x2100F000, len: 0x1000
Read from 0x2001E000, len: 0x1000


*****************************************
**** Compare Test <memory_read_byte> ****
*****************************************
```