Freescale MQX RTOS Example Guide

SPI example

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

The example

The example shows the usage of the SPI driver to perform the basic operation including reading data from, writing data to and erasing the memory of a SPI serial flash memory chip.

Running the example

The SPI application belongs to the set of examples of MQX using SPI driver. Hence the BSPCFG_ENABLE_SPIx macro must be set to non-zero in the user_config.h file prior to compilation of MQX libraries and the example itself.
What SPI port is used depends on the connection between the MCU board and the Memory board. This is normally SPI0 or SPI2.

To run the example the corresponding IDE, compiler, debugger and a terminal program are needed.
The MCU board and the Memory board are connected via the 4-wire SPI connection. The SPI_CLK, SPI_DOUT, SPI_DIN, SPI_CS pins are used.

Explaining the example

The application example creates only one task called main_task which is responsible for configuring the SPI protocol, erasing memory in memory board and reading data from and writing data to the memory board. The MCU is the master and the SPI serial flash memory is the slave in the SPI data transfer.

- In case the MCU board doesn't support slave chip select via SPI control register, the main_task manually configures one GPIO pin as CS pin and a CS callback function is provided:

```
#if BSP_SPI_MEMORY_GPIO_CS
   …..
  _mqx_int set_CS (uint32_t cs_mask, void *user_data)
  ….
   lwgpio_init(&spigpio, BSP_SPI_MEMORY_GPIO_CS,
                                    LWGPIO_DIR_OUTPUT,
   LWGPIO_VALUE_NOCHANGE)
   …..
   lwgpio_set_functionality(&spigpio,BSP_SPI_MUX_GPIO);
   …
   callback.CALLBACK = set_CS;
   …..
   ioctl (spifd, IO_IOCTL_SPI_SET_CS_CALLBACK, &callback)
  ….
#endif
```

- The ioctl() function is then called several times with different commands as input parameters to set up the attribute of the SPI protocol including the baudrate, the clock mode and the endianess of the slave device. The MCU is set up to be the master in the SPI protocol.

```
-------------- SPI driver example --------------

This example application demonstrates usage of SPI driver.
It transfers data to/from external memory over SPI bus.
The default settings in memory.h apply to TWR-MEM flash memory.
Current baud rate ... 10000000 Hz
Changing the baud rate to 500000 Hz ... OK
Current baud rate ... 468750 Hz
Setting clock mode to SPI_CLK_POL_PHA_MODE0 ... OK
Getting clock mode ... SPI_CLK_POL_PHA_MODE0
Setting endian to SPI_DEVICE_BIG_ENDIAN ... OK
Getting endian ... SPI_DEVICE_BIG_ENDIAN
Setting transfer mode to SPI_DEVICE_MASTER_MODE ... OK
Getting transfer mode ... SPI_DEVICE_MASTER_MODE
Clearing statistics ... not available, define BSPCFG_ENABLE_SPI_STATS
Getting statistics: not available, define BSPCFG_ENABLE_SPI_STATS
```

- The SPI serial flash memory is prepared to run the reading and writing test.
  - memory_read_status () returns the status of the serial flash memory
  - memory_set_protection () remove the write protection and memory_chip_erase() erase the serial flash memory.

```
Read memory status ... 0x1c
Enable write latch in memory ... OK
Read memory status ... 0x1e
Write unprotect memory ... OK
Enable write latch in memory ... OK
Read memory status ... 0x12
Write unprotect memory ... OK
Enable write latch in memory ... OK
Read memory status ... 0x12
Erase whole memory chip:
Read memory status ... 0x11
Read memory status ... 0x11
Read memory status ... 0x11
Read memory status ... 0x11
Read memory status ... 0x11
Read memory status ... 0x11
Read memory status ... 0x10
Erase chip ... OK
```

- Using functions memory_write_byte(), memory_read_byte(), memory_read_status() functions defined in the file spi_memory.c we can check the operation of serial flash with single data byte read and write process.

```
Enable write latch in memory ... OK
Read memory status ... 0x12
Write byte 0xba to location 0x000000f0 in memory ... done
Read memory status ... 0x10
Read byte from location 0x000000f0 in memory ... 0xba
Byte test ... OK
Getting statistics: not available, define BSPCFG_ENABLE_SPI_STATS
```

- Using function memory_write_data(), memory_read_data(), memory_read_status() functions defined in the file spi_memory.c we can check the operation of serial flash memory with read and write process for a sequence of data bytes.

```
Enable write latch in memory ... OK
Read memory status ... 0x12
Page write 12 bytes to location 0x000001f0 in memory:
Hello,World!
Read memory status ... 0x10
Reading 12 bytes from location 0x000001f0 in memory:
Hello,World!
Write short data test ... OK

Enable write latch in memory ... OK
Read memory status ... 0x12
Page write 16 bytes to location 0x000002f0 in memory:
ABCDEFGHIJKLMNOP
Enable write latch in memory ... OK
Read memory status ... 0x12
Page write 56 bytes to location 0x00000300 in memory:
QRSTUVWXYZ1234567890abcdefghijklmnopqrstuvwxyz1234567890
Read memory status ... 0x10
Reading 72 bytes from location 0x000002f0 in memory:
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890abcdefghijklmnopqrstuvwxyz1234567890
Write long data test ... OK
```

- The ioctl() function is then invoked with the command IO_IOCTL_SPI_READ_WRITE as input parameter to test the property of the SPI protocol in which the master and slave node exchange data simultaneously. Here the master writes the read command (0x03) and an address (0x0000f0) to the serial flash memory in addition to 6 dummy data bytes (0x00) to get the data in that address. The exchanged data in its SPI receive buffer is then displayed on the terminal.

```
IO_IOCTL_SPI_READ_WRITE ... OK
Simultaneous write and read - memory read from 0x000000f0 (10):
Write: 0x03 0x00 0x00 0xf0 0x00 0x00 0x00 0x00 0x00 0x00
Read : 0xff 0xff 0xff 0xff 0xba 0xff 0xff 0xff 0xff 0xff
Simultaneous read/write (data == 0xba) ... OK


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

- The main_task exits and example is finished.