Freescale Semiconductor
Application Note

Document Number:AN4791
Rev 0, 08/2013

Using DSPI as Slave on MQX™ 3.8

by: Terry Lv

Contents

1 Introduction

This application note describes the implementation of the Serial Peripheral Interface (SPI) module as slave on Freescale MQX RTOS 3.8. It is not recommended to make MCU act as an SPI slave. Currently, in all versions of the MQX release notes, slave mode is not supported.

This application note is for users who want to use DSPI as slave on the MCU.

This application note provides suggestions to make DSPI more stable and gives explanations on issues that might occur when DSPI acts in slave mode to the MCU.

2 SPI driver basic overview

There are two modes of SPI driver in MQX 3.8: Polling mode and DMA mode. Although both these modes can be used in SPI slave, the users should use Polling mode, because Polling mode is easier to control than DMA mode.

2.1 Polling mode overview

These are the driver files used in the Polling mode:

- Mqx\source\io\spi\polled\spi_pol.c

- Mqx\source\io\spi\polled\spi_pol_dspi.c
- Mqx\source\io\spi\polled\spi_pol\spi_pol_prv.h

Polling mode will transmit data out and loop for Transmit FIFO Fill Flag field of the Status Register (DSPI_SR[TFFF]) to be set, and then read data from POP RX FIFO (DSPI_POPR) register. Interrupt and FIFO is not used in this mode.

For more details, see *K60P100M100SF2V2RM: K60 Sub-Family Reference Manual* and *MQXIOUG: Freescale MQX ™ I/O Drivers - User's Guide*.

2.2 DMA mode overview

These are the driver files used in the Polling mode:

- Mqx\source\io\spi\int\spi_int_dspi.c
- Mqx\source\io\spi\int\spi_dma_dspi.c
- Mqx\source\io\spi\polled\spi_pol_dspi.c
- Mqx\source\io\spi\int\spi_int_prv.h

DMA mode transmits and receives data via DMA. Although it uses interrupt, FIFO is not used.

3 SPI slave mode overview

This figure shows the block diagram of an SPI module.

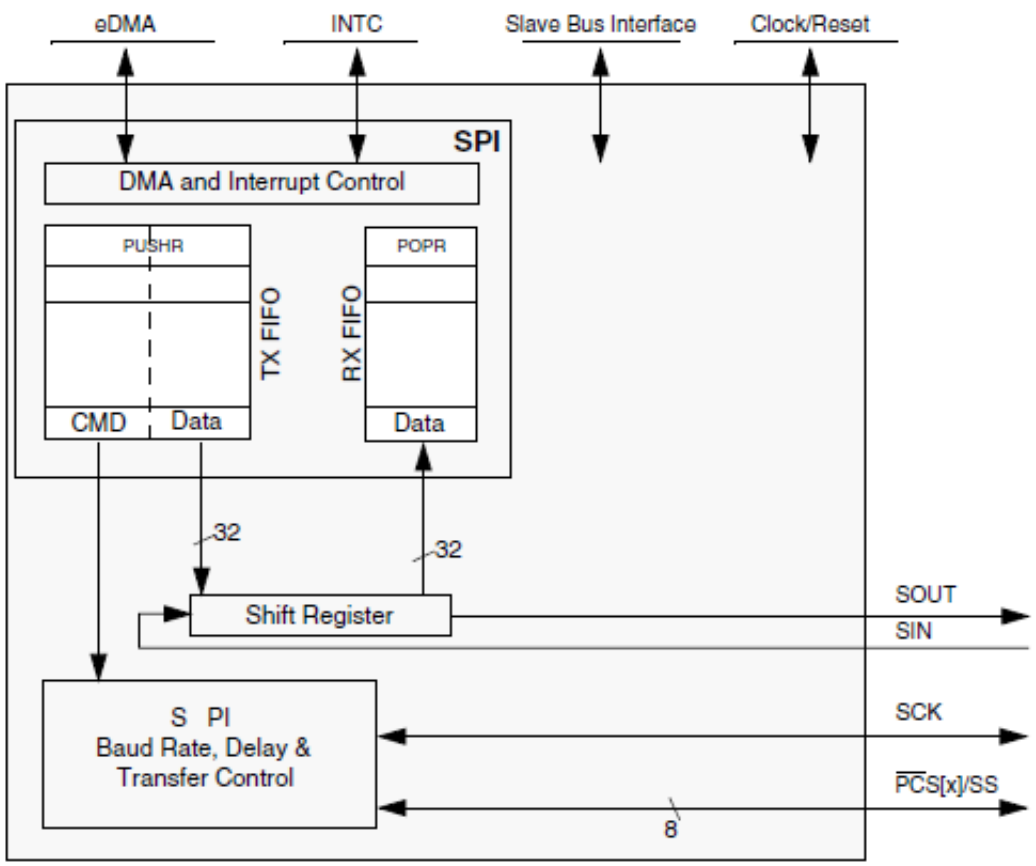

Figure 1. SPI block diagram

In SPI slave mode, the module responds to transfers initiated by an SPI bus master, but it does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase, and frame size must be set for successful communication with an SPI master.

SPI slave should have the same clock polarity, clock phase, and frame size as the master device.

Therefore, SPI slave device should always be ready for data transfer before the master device initiates a transfer. This means that SPI slave device should be fast enough, otherwise the slave may not be able to catch a transfer.

For slave mode in DSPI, it must be ensured that when the last entry in the TX FIFO buffer is completely transmitted, (that is, the corresponding Transfer Complete Flag (DSPI_SR[TCF]) is asserted and TX FIFO buffer is empty), the slave is deselected for any further serial communication, otherwise an underflow error occurs.

4 Program flow

For acting as a slave, SPI works in passive mode, which means that the data transfer and bit rate of SPI are controlled by the master device. The requirements for the master device are mentioned in the following section.

4.1 Master device requirement

To transfer data successfully and avoid an underflow in the slave mode, the master should fulfill these requirements.

- The master should try to run a bit slower than slave, to ensure that the slave is ready for each transfer.
- The master should pull up Slave Select (SS) signal and add a delay between each transfer. This delay time depends on different master devices. The least interval during testing has come out to be 32 ns.

4.2 Slave device program flow

4.2.1 Polling mode

Polling mode is simpler and easier to control than DMA mode, and thus it is the recommended mode for slave device.

Some mechanism can be used in polling mode such as sync command, or state machine, which can make it more stable. The SPI Slave application will try to get command from the master and do what the command requires. At last, it will inform the master that the command execution is done and it is ready to process other command.

This figure depicts a typical flow of a SPI Slave application.

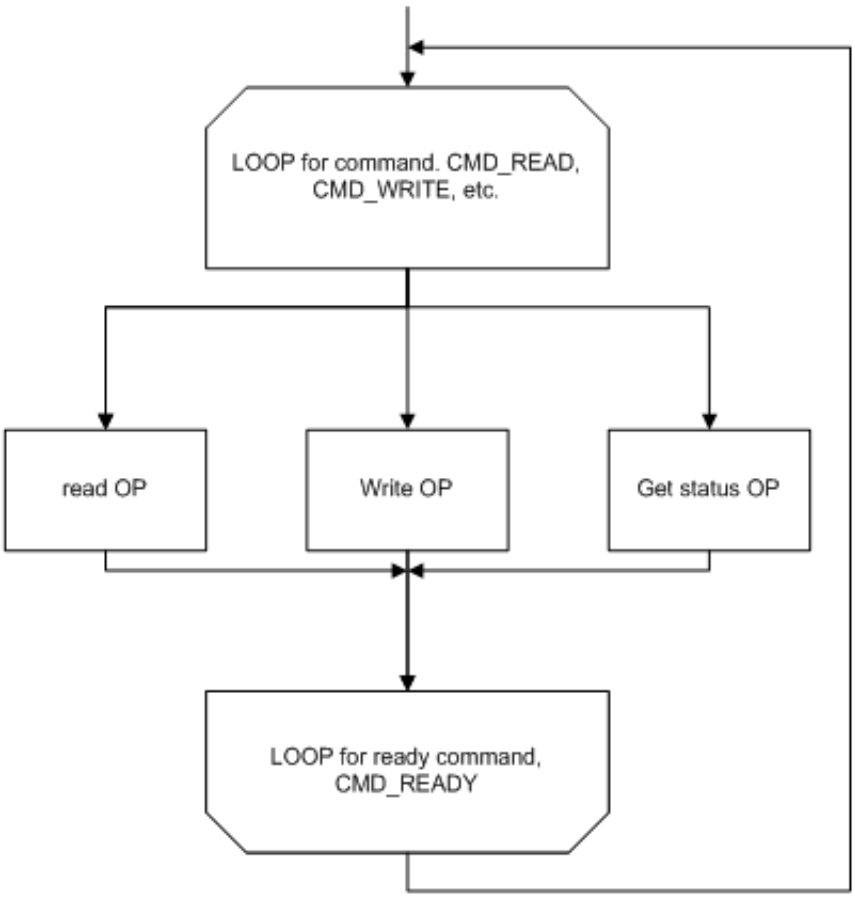

Figure 2. Typical SPI slave application work flow

These are some important considerations for Polling mode:

- Size of command is 1 byte. It is easier to poll for a 1-byte command.
- A state machine is optional.
- RX/TX FIFO is not used.

4.2.2 DMA mode

DMA mode is supposed to be faster than the Polling mode. Since DMA transfer in DMA mode is not controlled by software, the following workaround mechanism is used to avoid underflow in slave device.

1. Enlarge RX and TX buffer size by changing BSP_DSPI_RX_BUFFER_SIZE and BSP_DSPI_TX_BUFFER_SIZE.
2. When slave SPI needs to send response data, it's better to clear RX and TX buffer first.
3. If master needs to send data continuously, it's better to add a delay between each data package transfer.

5 SPI slave underflow explanation

As mentioned in SPI slave mode overview, for DSPI to work in slave mode, it must be ensured that when the last entry in the TX FIFO is completely transmitted (that is, the corresponding TCF flag is asserted and TXFIFO is empty), the slave is deselected for any further serial communication, otherwise, an underflow error occurs.

The underflow might be the frequent or common issue that occurs in DSPI slave device. It means that your slave device is not fast enough.

If data obtained from the slave device has some redundant bytes, there might be an underflow in the slave device.

For example:

Master should receive 0x8 0x0 0xE 0x0 from slave, but data received is 0xFF 0x8 0x0 0xE 0x0 or 0x00 0x8 0x0 0xE 0x0.

In a slave, every time a frame transmission begins, the next frame is popped from the TX FIFO and stored in a temporary register (to be transmitted next). If slave is not fast enough to put data to TX FIFO, the slave TX FIFO would be empty and there will be no more frames to transmit. A dummy data would then be transmitted and an overflow occurs.

6 Demo application code for polling mode

When using polling mode, the demo code will:

1. Polling for 1-byte command.
2. Parse 1-byte command.
3. Switch to process accordingly command.

   For example, if the command CMD_UPDATE_DATA_RESPONSE command, is received, switch to "case CMD_UPDATE_DATA_RESPONSE" to handle response data. Similarly, if the command CMD_UPDATE_DATA, is received, switch to "case CMD_UPDATE_DATA" to receive data from the master.

4. Send ready command back.

```
while (1) {

              uint_8 got_cmd = 0, send_cmd = 0;
              /* In MQX 3.8, fread() in spi transfer will block in reading data by checking
SR_RFDF bit. If SR_RFDF is set, which means master has initiated a transfer and data
arrived, the function will exchange data with master. */

              result = fread(&got_cmd, 1, 1, spifd); /* Polling to read 1 byte CMD */

              if (1 != result) {
                        printf("Failed to get CMD: %d\n", result);
                        goto err_out;

            }

            cmd_type = got_cmd;

            switch (got_cmd) {

            case CMD_UPDATE_DATA_RESPONSE:

                  /* Got Update data CMD */

                  result = fread(&response, 1, 3, spifd);

                  if (3 != result) {

                            printf("Failed to receive data: %d\n", result);

                            goto err_out;

                  }

                  cmd_type = got_cmd;

                  data_len = response[2]; /* Len for data received */

                  break;
```

```
            case CMD_UPDATE_DATA:

                  result = fread(receive_buffer, 1, data_len, spifd);

                  if (data_len != result) {

                           printf("Failed to receive data: %d\n", result);

                           goto err_out;
                }

                break;

            case SPI_CMD_UPDATE_READY:

                /* Send READY command back, this is a status-like command */

                send_cmd = SPI_CMD_UPDATE_READY;

                /* fwrite() in spi transfer will also block in reading data by checking
SR_RFDF bit. If SR_RFDF is set, which means master has initiated a transfer and data
arrived, the function will exchange data with master. */

                 result = fwrite(&send_cmd, 1, 1, spifd);

                 if (4 != result) {

                           printf("Failed to send ready cmd: %d\n", result);

                           goto err_out;

                 }

                 break;

          default:

                 break;
           }
}
```

7 Conclusion

This application note explains how to use DSPI as slave and mentions several considerations, while implementing the DSPI as a slave such as mode selection, timing, underflow error, and others. Generally, it is not recommended to use DSPI as SPI slave on the MCU, but if you need to use it, see this document for reference.

8 Reference

For additional information, see the following documents on freescale.com.

- *K60P100M100SF2RM: K60 Sub-Family Reference Manual*
- *MQXIOUG: Freescale MQX ™ I/O Drivers - User's Guide*

9 Revision history

Revision number	Date	Substantive changes
0	08/2013	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number Document Number AN4791
Revision 0, 08/2013