XMOS SPI Component

Document Number: X3171A

Publication Date: 2012/3/14

CONTENTS

1 Overview

IN THIS CHAPTER

1.1 Features

- SPI master and slave components.
- Simple API with functions to initialize, read and write values of 1, 2 or 4 bytes, or arbitrary length arrays, and shutdown.
- This component API is function-based and does not require a dedicated thread.
- Includes data transfer test code.

1.2 SPI modes

Mode	CPOL	CPHA	Supported
0	0	0	Yes
1	0	1	Yes
2	1	0	Yes
3	1	1	Yes

1.3 Memory requirements

Module	Minimal usage (Bytes)	Full usage (Bytes)
module_spi_master	2012	2640
module_spi_slave	1700	2028

1.4 Performance

1.4.1 Maximum SCLK frequencies

Mode	Master SCLK freq	Slave SCLK freq
0	25MHz	862kHz
1	25MHz	25MHz[1]
2	25MHz	862kHz
3	25MHz	25Mhz [1]

1.4.2 Bandwidth

1.4.2.1 SPI master

Mode	Master - 100MIPS thread (R/W)	Master - 50MIPS thread (R/W)
0	15.3Mbps / 4.2Mbps	11.3Mbps / 2.6Mbps
1	TBD	TBD
2	15.3Mbps / 4.2Mbps	11.3Mbps / 2.6Mbps
3	15.3Mbps / 15.6Mbps	11.4Mbps / 11.8Mbps

1.4.2.2 SPI slave

Mode	Slave - 100MIPS thread (R/W)	Slave - 50MIPS thread (R/W)
0	0.7Mbps / TBD	0.7Mbps / TBD
1	15.9Mbps / 15.2Mbps	4.8Mbps / 4.8Mbps with 6.25MHz SCLK
2	0.7Mbps / TBD	0.7Mbps / TBD
3	15.9Mbps / 15.2Mbps	4.8Mbps / 4.8Mbps with 6.25MHz SCLK

All bandwidth tests were conducted with the maximum supported SCLK frequency (see §1.4.1), unless otherwise stated. Where SCLK frequencies are given, these were found to be the maximum with which the tests could be completed correctly.

2 Hardware Platforms

IN THIS CHAPTER

Example applications are provided to test and demonstrate both the SPI master and slave modules. All provided demo applications are designed to run on the XK-1A Development Kit[2].

2.1 app_spi_loopback_demo

To run this application jumpers must first be placed across the following pins to connect the master and slave:

Master pin	Slave pin	SPI signal
XD0	XD1	MOSI
XD10	XD11	MISO
XD12	XD13	SCLK
XD22	XD23	SS

For information on the pin layout of the XK-1A please refer to the XK-1A Hardware Manual[3].

2.2 app_spi_master_demo

The application app_spi_master_demo can be run on the board without any modification.

[2] http://www.xmos.com/products/development-kits/xk-1a
[3] http://www.xmos.com/published/xk1ahw

3 Programming Guide

IN THIS CHAPTER

This section provides information on how to program applications using the SPI master and slave components.

3.1 Source code structure

3.1.1 Directory Structure

A typical SPI application will have at least three top level directories. The application will be contained in a directory starting with `app_`, the spi component source is in the `module_spi_master|slave` directory and the directory `module_xcommon` contains files required to build the application.

```
app_[my_app_name]/
module_spi_master|slave/
module_xcommon/
```

Of course the application may use other modules which can also be directories at this level. Which modules are compiled into the application is controlled by the `USED_MODULES` define in the application Makefile.

3.1.2 Key Files

The following header files contain prototypes of all functions required to use use the SPI components. The API is described in §4.

Figure 1: Key Files

File	Description
`spi_master.h`	SPI master API header file
`spi_slave.h`	SPI slave API header file

3.2 Demo Applications

This tutorial describes the demo applications included in the XMOS SPI package. §2 describes the required hardware setups to run the demos. The source for both demos can be found in the top level directory of the sc_spi component.

A basic knowledge of XC programming is assumed. For information on XMOS programming, you can find reference material about XC programming at the XMOS website[4].

To write an SPI enabled application for an XMOS device requires several things:

1. Write a Makefile for our application
2. Provide an spi_conf.h configuration file
3. Write the application code that uses the component

This process is described for both demos in the following sections.

3.2.1 app_spi_master_demo

This application uses module_spi_master to read and write data to the SPI slave flash device on the board.

Note: Running this program will overwrite any existing data in the flash.

3.2.1.1 Makefile

The Makefile is found in the top level directory of the application. It uses the general XMOS makefile in module_xcommon which compiles all the source files in the application and the modules that the application uses. We only have to add a couple of configuration options.

Firstly, this application is for an XK-1 development board so the TARGET variable needs to be set in the Makefile:

```
# The TARGET variable determines what target system the application is
# compiled for. It either refers to an XN file in the source directories
# or a valid argument for the --target option when compiling.
TARGET = XK-1
```

Secondly, the application name must be set:

```
# The APP_NAME variable determines the name of the final .xe file. It
   ↪ should
# not include the .xe postfix. If left blank the name will default to
# the project name
APP_NAME = app_spi_master_demo
```

Finally, the application will use the SPI master module. We state that the application uses this as follows:

```
# The USED_MODULES variable lists other module used by the application.
USED_MODULES = module_spi_master
```

[4]http://www.xmos.com/support/documentation

3.2.1.2 spi_conf.h

The spi_conf.h file is found in the src/ directory of the application. This file contains a series of `#defines` that configure the SPI component. The possible `#defines` that can be set are described in §4.1.

If not set, default values in the spi_master header file will be used.

Within this application we set the SPI mode to 3, and the default SPI clock divider to 2 (giving a SPI clock frequency of 25MHz).

```
// Copyright (c) 2011, XMOS Ltd., All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#define DEFAULT_SPI_CLOCK_DIV 2

#define SPI_MASTER_MODE 3
```

The flash chip on the XK-1 board supports SPI modes 0 and 3, and clock frequencies up to 104MHz. The demo can be recompiled and run in either SPI mode, and larger clock dividers.

3.2.1.3 Top level program structure

To make use of the SPI master module app_spi_master_demo must contain the following line:

```
#include "spi_master.h"
```

This application is contained in spi_master_demo.xc. Within this file is the `main()` function which sets up the SPI interface, and then calls the `read_jedec_id()`, `write_speed_test()`, and `read_speed_test()` functions to test the interface.

The `read_jedec_id()` function reads the device ID register and checks the returned value is as correct, outputting both the expected and actual values.

The `write_speed_test()` function writes a page of data to the flash, and then reads it back to ensure it was written correctly. The time taken to write the data is output.

The `read_speed_test()` function reads 50kB of data from the flash into a buffer. The time taken to complete the read is output.

3.2.1.4 Running the application

1. Run `xmake` within the app_spi_master_demo/ directory, to compile the program
2. Connect the XK-1 to your PC using an XTAG-2

3. Run `xrun --io bin/app_spi_master_demo.xe` to start the demo

3.2.2 app_spi_loopback_demo

This application uses both module_spi_master and module_spi_slave to create a loopback between 2 threads.

Note: Jumpers are required on the board to run this demo, as described in §2.

3.2.2.1 Makefile

The Makefile is found in the top level directory of the application. It uses the general XMOS makefile in module_xcommon which compiles all the source files in the application and the modules that the application uses. We only have to add a couple of configuration options.

Firstly, this application is for an XK-1 development board so the TARGET variable needs to be set in the Makefile:

```
# The TARGET variable determines what target system the application is
# compiled for. It either refers to an XN file in the source directories
# or a valid argument for the --target option when compiling.
TARGET = XK-1
```

Secondly, the application name must be set:

```
# The APP_NAME variable determines the name of the final .xe file. It
   ↪ should
# not include the .xe postfix. If left blank the name will default to
# the project name
APP_NAME = app_spi_loopback_demo
```

Finally, the application will use both the SPI master and slave modules. We state that the application uses them as follows:

```
# The USED_MODULES variable lists other module used by the application.
USED_MODULES = module_spi_master module_spi_slave
```

3.2.2.2 spi_conf.h

The spi_conf.h file is found in the src/ directory of the application. This file contains a series of `#defines` that configure the SPI component. The possible `#defines` that can be set are described in §4.1.

If not set, default values in the spi_master and spi_slave header files will be used.

As both master and slave modules can run in all the SPI modes, within this application any mode can be selected. Changing the `#define SPI_MODE` ensures that both master and slave are run in the same mode as one another.

The maximum frequency sclk supported by the slave module differs for each SPI mode, `#define DEFAULT_SPI_CLOCK_DIV` sets the highest frequency that will work in any SPI mode. `#define SPI_CLOCK_DIV` sets the frequency actually used by the application.

```
// Copyright (c) 2011, XMOS Ltd., All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#define DEFAULT_SPI_CLOCK_DIV 58 // Maximum frequency suitable for SPI
  ↪ slave in any SPI mode

#define SPI_CLOCK_DIV 2 //DEFAULT_SPI_CLOCK_DIV

#define SPI_MODE 3
// Ensure that master and slave always operate in the same mode
#define SPI_MASTER_MODE SPI_MODE
#define SPI_SLAVE_MODE SPI_MODE
```

The application comes configured to run in SPI mode 3, with an SPI clock divider of 2 (giving a SPI clock frequency of 25MHz). The demo can be recompiled and run in other SPI modes, with different clock dividers.

3.2.2.3 Top level program structure

To make use of the SPI master and slave modules app_spi_loopback_demo must contain the following lines:

```
#include "spi_master.h"
#include "spi_slave.h"
```

This application is contained in spi_loopback_demo.xc. Within this file is the `main()` function which starts the SPI master and slave running in parallel threads by calling the functions `run_master()` and `run_slave()`. These run functions initialise the interface modules and run the demo a set number of times. The `master_demo()` function checks that the data returned from the slave is correct. The interfaces are shut down after the final run of the demo.

3.2.2.4 Running the application

1. Run `xmake` within the app_spi_loopback_demo/ directory, to compile the program
2. Connect the XK-1 to your PC using an XTAG-2

3. Run `xrun --io bin/app_spi_loopback_demo.xe` to start the demo

4 API

4.1 Configuration Defines

The file spi_conf.h can be provided in the application source code, without it the default values specified in spi_master.h and spi_slave.h will be used. This file can set the following defines:

4.1.1 SPI master Defines

DEFAULT_SPI_CLOCK_DIV

This define sets the default clock divider, which the application can use when initialising the SPI master. See **spi_clock_div** parameter of `spi_master_init()` in §4.2.2 for clock divider format.

SPI_MASTER_MODE

The SPI mode the master operates in.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

4.1.2 SPI slave Defines

SPI_SLAVE_MODE

The SPI mode the slave operates in.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

4.2 SPI master API

4.2.1 Data Structures

`spi_master_interface`

Structure containing the resources required for the SPI master interface.

It consists of two 8bit buffered output ports, and one 8bit input port.

Select lines are intentionally not part of API, they are simple port outputs, which depend on how many slaves there are and how they're connected.

This structure has the following members:

`clock blk1`

`clock blk2`

`out buffered port:8 mosi`

`out buffered port:8 sclk`

`in buffered port:8 miso`

4.2.2 Configuration Functions

`void spi_master_init(spi_master_interface &spi_if, int spi_clock_div)`

Configure ports and clocks, clearing port buffers.

Must be called before any SPI data input or output functions are used.

Example: To achieve an sclk frequency of 25MHz, a divider of 2 must be specified, as 100(MHz)/(2*2) = 25(MHz).

Example: To achieve an sclk frequency of 625kHz, a divider of 80 must be specified, as 100(MHz)/(2*80) = 0.625(MHz).

This function has the following parameters:

`spi_if`	Resources for the SPI interface being initialised

`spi_clock_div`
: SPI clock frequency is fref/(2*spi_clock_div), where freq defaults to 100MHz

`void spi_master_shutdown(spi_master_interface &spi_if)`

Stops the clocks running.

Should be called when all SPI input and output is completed.

This function has the following parameters:

`spi_if`	Resources for the SPI interface being shutdown

4.2.3 Receive Functions

`unsigned char spi_master_in_byte(spi_master_interface &spi_if)`

Receive one byte.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received byte

`unsigned short spi_master_in_short(spi_master_interface &spi_if)`

Receive one short.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received short

`unsigned int spi_master_in_word(spi_master_interface &spi_if)`

Receive one word.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received word

```
void spi_master_in_buffer(spi_master_interface &spi_if,
                          unsigned char buffer[],
                          int num_bytes)
```

Receive specified number of bytes.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`buffer`	The array the received data will be written to
`num_bytes`	The number of bytes to read from the SPI interface, this must not be greater than the size of buffer

4.2.4 Transmit Functions

```
void spi_master_out_byte(spi_master_interface &spi_if, unsigned char data)
```

Transmit one byte.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`data`	The byte to transmit

```
void spi_master_out_short(spi_master_interface &spi_if,
                          unsigned short data)
```

Transmit one short.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`data`	The short to transmit

```
void spi_master_out_word(spi_master_interface &spi_if, unsigned int data)
```

Transmit one word.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

`data`	The word to transmit

```
void spi_master_out_buffer(spi_master_interface &spi_if,
                           const unsigned char buffer[],
                           int num_bytes)
```

Transmit specified number of bytes.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`buffer`	The array of data to transmit
`num_bytes`	The number of bytes to write to the SPI interface, this must not be greater than the size of buffer

4.3 SPI slave API

4.3.1 Data Structures

`spi_slave_interface`

Structure containing the resources required for the SPI slave interface.

It consists of two 1bit input ports, one 8bit buffered input port, and one 8bit buffered output port.

This structure has the following members:

```
clock blk

in port ss

in buffered port:8 mosi

out buffered port:8 miso

in port sclk
```

4.3.2 Configuration Functions

```
void spi_slave_init(spi_slave_interface &spi_if)
```

Configure ports and clocks, clearing port buffers.

Must be called before any SPI data input or output functions are used.

This function has the following parameters:

`spi_if`	Resources for the SPI interface being initialised

`void spi_slave_shutdown(spi_slave_interface &spi_if)`

Stops the clocks running, and disables the ports.

Should be called when all SPI input and output is completed.

This function has the following parameters:

`spi_if`	Resources for the SPI interface being shutdown

4.3.3 Receive Functions

`unsigned char spi_slave_in_byte(spi_slave_interface &spi_if)`

Receive one byte.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received byte

`unsigned short spi_slave_in_short(spi_slave_interface &spi_if)`

Receive one short.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received short

`unsigned int spi_slave_in_word(spi_slave_interface &spi_if)`

Receive one word.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface

This function returns:

The received word

```
void spi_slave_in_buffer(spi_slave_interface &spi_if,
                         unsigned char buffer[],
                         int num_bytes)
```

Receive specified number of bytes.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`buffer`	The array the received data will be written to
`num_bytes`	The number of bytes to read from the SPI interface, this must not be greater than the size of buffer

4.3.4 Transmit Functions

```
void spi_slave_out_byte(spi_slave_interface &spi_if, unsigned char data)
```

Transmit one byte.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`data`	The byte to transmit

```
void spi_slave_out_short(spi_slave_interface &spi_if, unsigned short data)
```

Transmit one short.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`data`	The short to transmit

```
void spi_slave_out_word(spi_slave_interface &spi_if, unsigned int data)
```

Transmit one word.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`data`	The word to transmit

```
void spi_slave_out_buffer(spi_slave_interface &spi_if,
                          const unsigned char buffer[],
                          int num_bytes)
```

Transmit specified number of bytes.

Most significant bit first order. Big endian byte order.

This function has the following parameters:

`spi_if`	Resources for the SPI interface
`buffer`	The array of data to transmit
`num_bytes`	The number of bytes to write to the SPI interface, this must not be greater than the size of buffer

Table of Contents

X3171A