SPI multifield HDL core

Carlos Gil Soriano
BE-CO-HT
carlos.gil.soriano@cern.ch

July 20, 2012

Abstract

A configurable SPI multifield HDL core is shown. This core is able to specify three kind of fields to be sent according to SPI communication and been able to configure independently the lenght of every field.

The core is targeted for complexer uses of SPI communications, like writing blocks of EEPROM memories which typically requiere three fields.

The following subjects are addressed:

- The registers to control the module.
- Step-by-step instructions for proper use. access.

Revision history		
HDL version	**Module**	**Date**
0.1	SPI master multifield	July 20, 2012

Contents

1 Structure

The SPI module contains several blocks related the following way:

– spi_master_top.vhd
—— spi_master_regs.vhd
—— spi_master_slave_core.vhd
——— FIFO_dispatcher.vhd
——— gc_counter.vhd
——— gc_clk_divider.vhd

The top module combines two components: *spi_master_core* and *spi_master_regs*. The first one can be used independently from thei the top module, saving some interconnection lines and allowing a direct way of using the module. If access to the control registers *SPI[X]* through classic Wishbone interface is desired, then the top module must be used.

Due to the target use of this SPI core (block transfers for memory interfaces) all the three input fields are offered in both the top and the core modules.

Internally, the data in every of the three set of fields is registered by the control registers, either by directly writing into the *SPI[X]* register (in the case of *spi_master_core*) or through wishbone (*spi_master_top*).

1.1 Dependencies

Three components used in this core belongs to general use in CTDAH board. Due to that they are packed inside **ctdah_lib**. The required components to be imported are:

– FIFO_dispatcher.vhd
– gc_counter.vhd
– gc_clk_divider.vhd

2 Registers

2.1 SPI0

The SPI0 is a write-read register.

Bits	Field	Meaning	Default
0	**CPOL**	Clock POLarity when idle	**"00000"**
1	**CPHA**	Clock PHAse	**"00000"**
4-2	x	Reserved	**"00000"**
13-5	**BDATA**	Bytes of DATA to be sent	**c_INST_LENGTH**
22-14	**BADDR**	Bytes of ADDRess to be sent	**c_ADDR_LENGTH**
31-23	**BINST**	Bytes of INSTruction to be sent	**c_DATA_LENGTH**

2.2 SPI1

The SPI1 is a write-read register.

Bits	Field	Meaning	Default
0	**PUSH_DATA**	PUSH DATA bytes into internal SPI core memory	**'0'**
1	**PUSH_ADDR**	PUSH ADDRess bytes into internal SPI core memory	**'0'**
2	**PUSH_INST**	PUSH INSTruction bytes into internal SPI core memory	**'0'**
5-3	x	Reserved	**"000"**
6	**SEND_DATA**	DATA bytes will be sent in a write operation	**'0'**
7	**SEND_ADDR**	ADDR bytes will be sent in a write operation	**'0'**
8	**SEND_INST**	INST bytes will be sent in a write operation	**'0'**
9	**SEND_OP**	perform a SEND OPeration	**'0'**
11-10	y	Reserved	**"00"**
15-12	**CLK_DIV**	CLocK DIVider	**X"0"**
31-16	z	Reserved	**X"0000"**

2.3 SPI2

The SPI2 register is a read-only register.

Bits	Field	Meaning	Default
0	**SENT_DATA**	DATA was SENT	**'0'**
1	**SENT_ADDR**	ADDRess was SENT	**'0'**
2	**SENT_INST**	INSTruction was SENT	**'0'**
3	**SENT_OP**	OPeration was SENT	**'0'**
11-4	x	Reserved	**X"00"**
15-12	**CLK_DIV**	CLocK DIVision	**X"0"**

3 Internal memory mapping

The internal registers map is as follow:

Address	Register	Access
0x0	*SPI0*	Write-read
0x1	*SPI1*	Write-read
0x2	*SPI2*	Read-only

4 How to use it

4.1 Outputting a stream of data through SPI

1. Specify the divider it will be used to construct the SPI clk signal out of the general clk signal.
2. Place the values to be sent in *inst_i*, *addr_i* and *data_i*.
3. Register their values by one-clock asserting *PUSH_[X]* bits in *SPI1* register.
4. Specify which fields must be sent by asserting *SEND_[X]* bits in *SPI1* register. This bits must keep asserted during the whole send operation.
5. Specify the length in bytes of every field by writing *SPI0*.
6. Assert *SEND_OP* bit field in *SPI1* register. It must be keep asserted until *SENT_OP* is received.