Key Design Features

- Synthesizable, technology independent IP Core for FPGA, ASIC and SoC
- Supplied as human readable VHDL (or Verilog) source code
- SPI serial-bus compliant
- Supports up to 16 slave devices
- Intuitive command interface featuring a simple valid-ready handshake protocol
- Input/output FIFOs permit queuing of sequential SPI requests and corresponding read data
- Architecture allows sustained 8-bit read/write operations
- Configurable serial clock frequency
- Configurable clock polarity setting (CPOL)
- Configurable clock phase setting (CPHA)
- Capable of full-duplex or half-duplex operation
- Data rates of up to 40 Mbps+[1]

Applications

- Driving SPI slave devices
- Inter-chip board-level communications
- Robust communication at higher data rates than other serial protocols such as I2C, UART and USB 1.0

Generic Parameters

Generic name	***Description***	***Type***	***Valid range***
t_period	Serial clock period (in system clock cycles)	integer	≥ 5
cpol	Clock polarity	integer	0, 1 (As per SPI specification)
cpha	Clock phase	integer	0, 1 (As per SPI specification)
wfifo_depth	Master instruction write FIFO depth	integer	≥ 2
wfifo_depth_log2	Master instruction write FIFO depth log2	integer	log2 (wfifo_depth)
rfifo_depth	Slave read data FIFO depth	integer	≥ 2
rfifo_depth_log2	Slave read data FIFO depth log2	integer	log2 (rfifo_depth)

Block Diagram

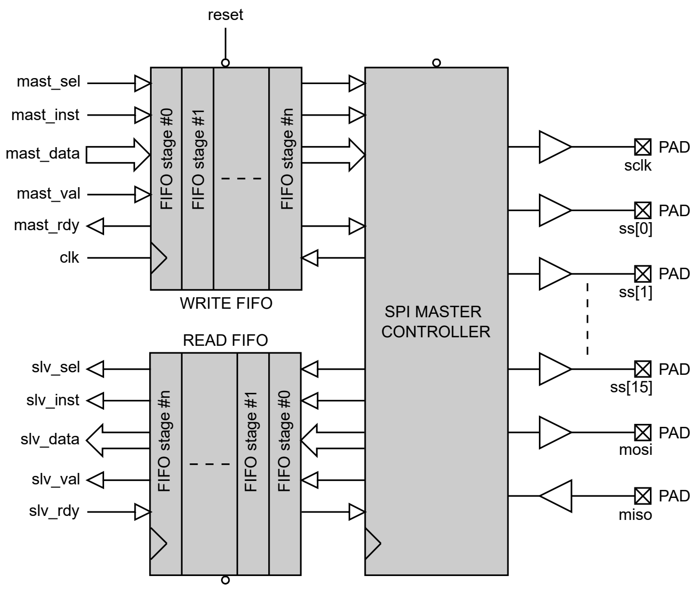

Figure 1: SPI Master Interface Controller architecture

Pin-out Description

Pin name	***I/O***	***Description***	***Active state***
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
mast_sel[3:0]	in	Slave device selection (0 to 15)	data
mast_inst[1:0]	in	Master instruction	data
mast_data[7:0]	in	Master data to be serialized	data
mast_val	in	Master instruction valid	high
mast_rdy	out	Master ready handshake	high
sclk	out	SPI serial clock	rising or falling edge[2]
ss[15:0]	out	SPI slave select (One hot decode)	low
mosi	out	SPI Master out / Slave in	data
miso	in	SPI Master in / Slave out	data
slv_sel[3:0]	out	Slave device id	data
slv_inst[1:0]	out	Slave instruction	data
slv_data[7:0]	out	Slave data received from slave device	data
slv_val	out	Slave data valid	high
slv_rdy	in	Slave ready handshake	high

1 Maximum attainable data rate will generally be determined by the slave SPI device and the physical characteristics of the bus

2 Note that the serial clock characteristics are dependent on the CPOL and CPHA settings. See the SPI specification for more details

General Description

The SPI_MASTER IP Core is an SPI compliant serial interface controller capable of driving up to 16 different slave devices in full-duplex operation. The controller receives data and instructions via the master instruction interface. These instructions are then processed by the controller core in order to generate the appropriate signals on the SPI bus. The serial slave data on the SPI bus is also captured by the controller and de-serialized for presentation at the slave read data port.

The SPI master controller is comprised of three main blocks as described by Figure 1. These blocks are the master instruction write FIFO, the SPI controller core and the slave read-data output FIFO.

The serial clock-period is determined by the generic parameter *t_period.* This parameter specifies the *sclk* period in system clock cycles. As an example, if the system clock '*clk*' is running at 130 MHz and a serial clock frequency of 10 MHz is required, a value of *t_period* = 13 should be specified. In addition, the generic parameters *cpol* and *cpha* permit the clock polarity and phase characteristics to be specified as per the SPI specification. The table below shows a brief summary of these settings.

CPOL	CPHA	Description
0	0	Serial clock default state logic '0' Data sampled on rising-edge of serial clock Data changes on falling-edge of serial clock
0	1	Serial clock default state logic '0' Data sampled on falling-edge of serial clock Data changes on rising-edge of serial clock
1	0	Serial clock default state logic '1' Data sampled on falling-edge of serial clock Data changes on rising-edge of serial clock
1	1	Serial clock default state logic '1' Data sampled on rising-edge of serial clock Data changes on falling-edge of serial clock

Master Write FIFO

Instructions to the SPI master controller are sent via an input FIFO whose depth is determined by the generic parameter *wfifo_depth*. The write FIFO interface operates in accordance with a simple valid/ready pipeline protocol meaning that instructions and data are written to the FIFO on the rising edge of *clk* when *mast_val* is high and *mast_rdy* is high[3]

The write FIFO may be used to 'queue up' a sequence of commands and data while current commands are being processed on the bus. As soon as the write FIFO becomes full then the FIFO will disable the *mast_rdy* signal signifying that further requests are not possible.

Likewise, the *mast_rdy* signal will also be disabled if the slave read-data FIFO becomes full. In both situations, no further commands will be accepted by the SPI controller until the FIFOs have emptied.

The instructions to the SPI controller are very intuitive and follow the exact sequence of commands that the user wishes to appear on the SPI bus. The following table outlines the set of commands accepted by the controller via the master write FIFO.

MASTER INSTRUCTION INPUT FORMAT

mast_inst[3:0]	*mast_data[7:0]*	*Description*
"00"	[7:0]: Write data	SPI WRITE Write 8-bits serially on the SPI bus. Ignore the read data bits (Half-duplex operation)
"01"	[7:0]: 'X' Don't care	SPI READ Read 8-bits serially on the SPI bus. Write data is don't care. (Half-duplex operation)
"10"	[7:0]: Write data	SPI READ/WRITE Read and write 8-bits serially on the SPI bus. (Full-duplex operation)
"11"	[7:0]: Don't care	NULL Dummy instruction. (May be used to force the slave select signal inactive between sequential SPI transfers)

As an example, to write two consecutive bytes followed by two consecutive reads from the same slave device with the slave select signal forced inactive-high in between, the instructions "00", "00", "11", "01", "01" would be sent.

Of course, the exact sequence of instructions required will depend on the functionality of the slave device that is to be accessed. For this reason, there is no restriction in the ordering of instructions that may be sent to to the SPI master controller.

SPI Master Controller Core

The master controller is a state-machine that accepts instructions from the write FIFO and generates the appropriate signals on the SPI bus. Immediately after an asynchronous reset of the core, the state machine starts in the reset state with the slave select lines '*ss*' inactive high. On receipt of the first valid instruction, the state machine will take control of the bus and drive the *sclk*, *ss* and *mosi* lines in response to the received instructions and data. If the instruction is a read operation, the controller will also sample the input serial data on the *miso* pin.

The default state of the serial clock and the edges in which the serial data are sampled and changed is dependent on the *cpol* and *cpha* settings as described earlier.

Slave Read FIFO

For every master instruction received by the controller, the controller also sends a copy of the original instruction plus the slave read data (if applicable) to the slave read FIFO. In the case that the originating instruction was a slave write then the slave read data contains a copy of the original master data. The following table gives a brief summary of the instruction format.

3 See Zipcores application note: app_note_zc001.pdf for more examples of the valid/ready protocol and it's implementation

SLAVE INSTRUCTION OUTPUT FORMAT

slv_inst[3:0]	***slv_data[7:0]***	***Description***
"00"	[7:0]: (same as original master data)	SPI WRITE
"01"	[7:0]: (slave read data)	SPI READ
"10"	[7:0]: (slave read data)	SPI READ/WRITE
"11"	[7:1]: slave address [0]: r/w flag	ADDR

Note that the *slv_inst* outputs are identical to the *mast_inst* inputs. The exception is the NULL instruction where nothing is logged in the slave read FIFO.

Functional Timing

Figure 2 shows a simple series of instructions sent to the controller. The sequence is: WRITE, READ, WRITE and READ/WRITE. Note that the FIFO is full after the third instruction and *mast_rdy* is de-asserted for one clock cycle. In the following cycle, *mast_rdy* goes high and the final instruction is transferred.

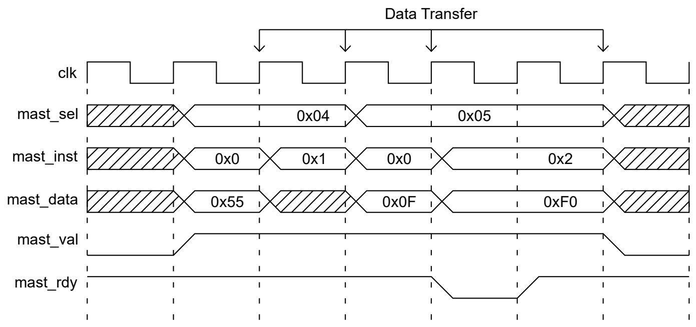

Figure 2: Master instruction interface timing

Figure 3 demonstrates the series of responses on the slave port for the same set of instructions.

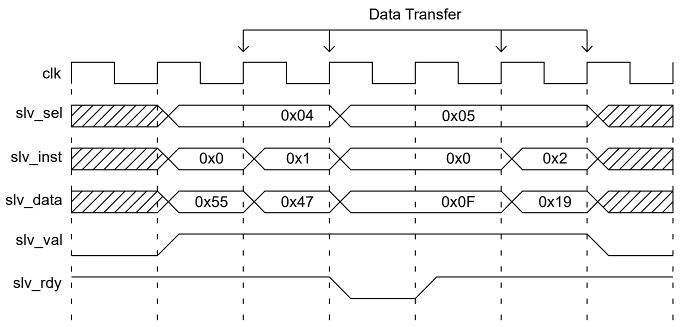

Figure 3: Slave instruction interface timing

Finally, Figure 4 demonstrates the corresponding SPI bus signals that are generated in response to the sequence of instructions described above.

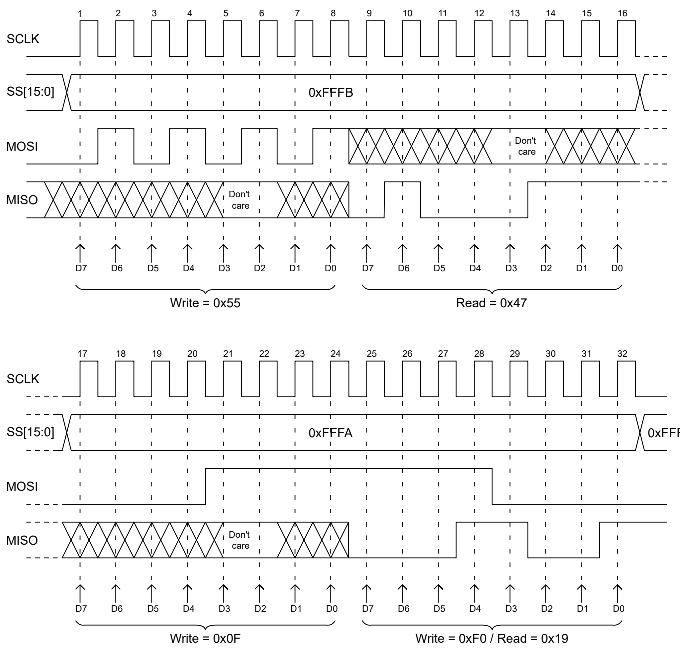

Figure 4: SPI bus signalling

Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	***Description***
spi_fifo.vhd	Input/output FIFOs
spi_master_cont.vhd	SPI master controller
spi_master.vhd	Top-level block
spi_master_bench.vhd	Top-level test bench

Functional Testing

An example VHDL test bench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. spi_fifo.vhd
2. spi_master_cont.vhd
3. spi_mast.vhd
4. spi_master_bench.vhd

The VHDL test bench instantiates the spi_master component in a loopback configuration – meaning that the SPI *mosi* pin is fed back to the SPI *miso* pin. In addition, the user may modify the generic parameters on the SPI master component in order to select the desired clock relationships (CPOL/CPHA) and the desired serial clock frequency.

In the default set up, the simulation must be run for around 100 ms during which time a random sequence of instructions will be sent to the master controller.

The simulation generates two text files: *spi_master_in.txt* and *spi_master_out.txt*. These files contain, respectively, the input and output data captured at the master instruction and slave data ports during the course of the test. The contents of these two files may be compared to verify the operation of the SPI master controller.

Development Board Testing

The SPI Master Interface Controller was implemented on a Xilinx® Spartan-6 FPGA running at a system clock frequency of 120 MHz. The SPI slave device was a Microchip® 12-bit serial SPI DAC part number MCP4822 running with a serial clock frequency of 20 MHz.

In all examples, the SPI mode was specified as mode 0,0 (CPOL = 0, CPHA = 0) meaning that data is sampled on the rising edge of the clock and changed on the falling edge. The default state of the clock is logic '0'.

Figure 5 below shows the relationship between the slave select signal and the serial clock running at 20 MHz. The top trace is *sclk* and the bottom trace is *ss[0]*. Notice that the slave select signal has a setup/hold time of at least 1/2 serial clock before and after the serial clock is enabled and disabled. This ensures compatibility with slower SPI slave devices.

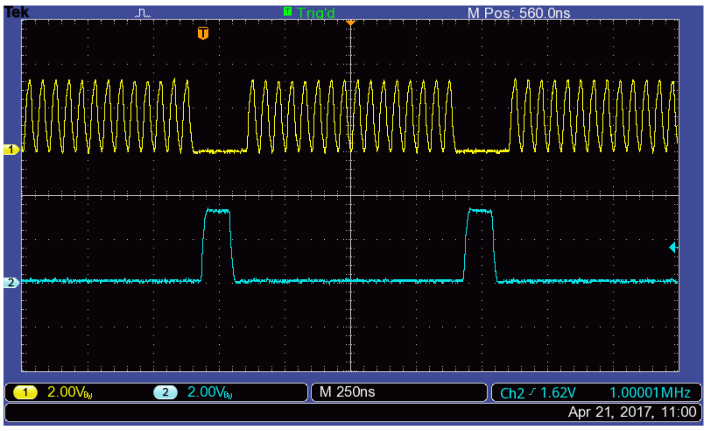

Figure 5: SPI serial clock/slave select relationship

The next figure demonstrates the corresponding 2-byte write at 20 MHz of the value 0xBF7D. The top trace is the serial clock *sclk* and the bottom trace is the serial data out *mosi*.

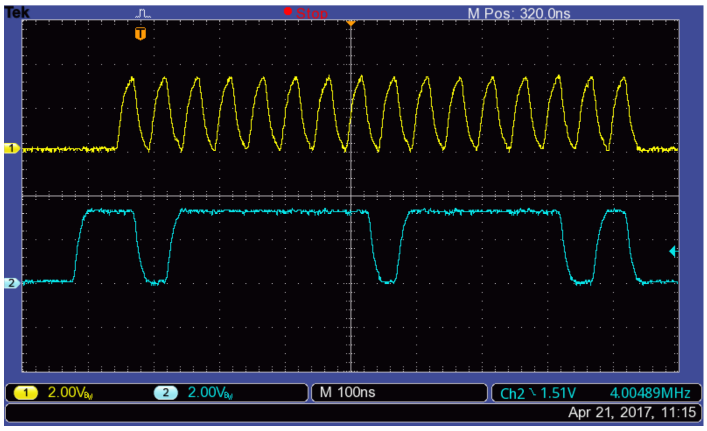

Figure 6: 16-bit SPI write detail at 20 MHz

Finally, Figure 7 shows detail of the output sine waves generated by the MCP4822 DAC component. A perfect 3.33 kHz sine wave was generated on each DAC output channel. The SPI writes were set up so that a sample was sent to each DAC channel in turn (A,B,A,B etc.) every 20th SPI clock (16 clocks then 4 wait clocks). This effectively meant that the DAC update rate was 1 MHz (or 500 kHz per channel).

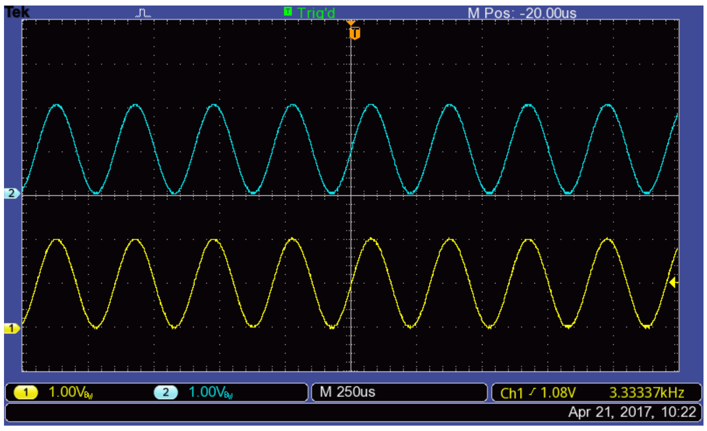

Figure 7: DAC A and B output signals from the MCP4822 - both sinusoids at 3.33 kHz