SPI Master / Slave Cores Specification

SPI_MASTER_SLAVE

SPI Master and Slave Interfaces

VHDL

RTL Architecture

Author: Jonny Doin

jdoin@opencores.org

Manual rev. 1.15

August 2, 2011

Revision History

Rev.	Date	Author	Description
0.1	11/05/18	JD	First Draft Described the SPI_MASTER and SPI_SLAVE cores.
0.97	11/06/12	JD	Added Design Considerations.
1.15	11/08/02	JD	Major redesign.

Contents

1 Introduction

The SPI_MASTER_SLAVE core implements two related but independent design blocks: the SPI_MASTER and the SPI_SLAVE blocks.

These core IPs are targeted for generic FPGA applications that need SPI communications from 8 bits to 64 bits of SPI word size, from < 1MHz of line frequency to > 50MHz.

Each core is a small RTL description for the widely used Serial Peripheral Interface, written in VHDL.

The SPI bus signals follow the de-facto standard for the SPI interface, and are named after the Motorola original naming convention, with the 4 SPI signals (SSEL, SCK, MOSI, MISO). The SPI mode and serial word size can be controlled at instantiation by VHDL generics.

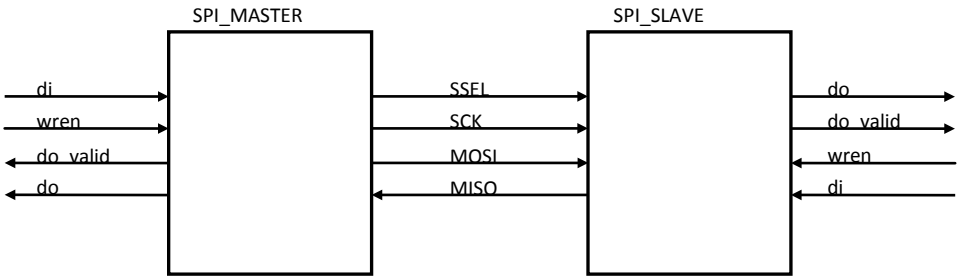


FIGURE 1 – Simplified SPI_MASTER and SPI_SLAVE cores showing the main ports only

There are two data interfaces to each core, the serial SPI bus interface, with the SPI signals, and the parallel read/write interface. Each data interface work on asynchronous clock domains, the serial interface clock, and the parallel interface clock. Cross-clock bridge logic takes care of the data transfer over the async domains, and guarantees glitch-free operation on data setup/hold times between the async clocks.

All operation is fully static, and the parallel interface is simple to use, similar to a synchronous RAM block.

Although the models are written in high-level, vendor-independent VHDL suitable to synthesis by standard VHDL synthesizers, the description follows the Xilinx recommendations for optimized Spartan-6 and newer CLB structures inference engines, such as reduced control sets and global initialization of registers using explicit initialization values at signal declarations. These are clean practices that do not yield obscure descriptions, and should not hinder the targeting to other technologies.

2

Design Considerations

Every IP block in a design has some degree of flexibility in the actual implementation of the intended functionality modeled. If the target technology is an FPGA, there are different design considerations and certain code practices to achieve best area, resource utilization, operating frequency, module integration, technology independency and verification strategy than if the target is an ASIC. The design goal is to describe functionality that is largely independent of implementation technology, but certain architectural aspects are more FPGA oriented, like fully synchronous design, and RTL description that fits an underlying logic block more closely presented by an FPGA fabric. The design considerations that follow are proper to FPGA synthesis, i.e., pipelined LUT/FF logic structures, with very few global clock lines.

The SPI master/slave IP follows some assumptions that governed the design:

- the primary use of the SPI interface is to communicate with peripherals on the range of 1MHz to 80MHz of SCK operating frequency;
- implementation should use only generic CLB resources, with no FPGA-specific hard macros or specialized block RAM structures to achieve the desired functionality;
- flexibility of SPI mode and word size selection is desirable, but also the model needs to have a very small footprint;
- synthesis should be possible with any reasonable synthesis tool, with little or no special constraint declaration;
- the code should be easily verifiable and high-level, but also correctly synthesizable for FPGA architectures with fair control over the inferred logic structures;

RTL CODE

With these constraints in mind, a description at the RTL level was chosen, with no vendor-specific structures or meta-commands, following generally recommended techniques for RTL state machine design.

Inference of the intended FFDs and LUT/FF logic layers is straightforward, following an explicit clock model with explicit combinatorial/registered pairs of signals declared in the model. This style leaves little room for ambiguous synthesis interpretation, while is still very easy to visualize the actual hardware functions described. On the other hand, RTL descriptions may be very cryptic when describing functionality at a higher abstract level. We use well commented code, with a data path centric approach to render the description as easy to envision as possible. The advantages of using RTL are evident for simple circuit blocks like the SPI_MASTER and SPI_SLAVE cores, which have a straightforward streaming data processing model. It is easy to visualize sequential logic pipelines and next-state logic.

CONFIGURATION VIA GENERICS

To achieve the least area possible, but still have a flexible IP, all configurations for SPI mode, word size and pipeline behavior are selected via VHDL generics at module instantiation, instead of dynamic reconfiguration. This preserves flexibility and usability, while achieving best area, by removing unnecessary logic that would be needed to implement runtime configuration.

The interface master/slave function is also selected by instantiating the appropriate model, instead of selecting the master/slave function at runtime. The two functions are similar but different enough to be implemented in separate models. This contributes for a better functional/area compromise.

VHDL CODE STYLE TO ACHIEVE BEST SYNTHESIS

Although the description is fairly vendor-independent, we chose Xilinx Spartan-6 technology as the target, and followed the accepted code style for correct inference of optimized Xilinx CLB logic, having the Spartan-6 slice limitations on the use of register initialization and control sets.

The Spartan-6 and newer (45nm and smaller) Xilinx technologies have severe limitations on the use of control sets, which are the flip-flop control signals: CLOCK, RESET/PRESET and CLOCK ENABLE. Each slice has 8 registers, and all 8

registers must share the same set of control signals, and all control signals except the clock must be positive logic. Further, there are limitations on initialization logic. A flip-flop on these newer Xilinx architectures cannot have both an async set and a reset function, and cannot have a global init value that is the opposite polarity of the async set/reset function. So, in addition to the good general practices for description of FPGA RTL circuits, to achieve good resource utilization on these new Xilinx devices, the following guidelines apply:

- use only global initialization whenever possible. This means declaring explicit initial values on the signal declarations, and use only 'zero' values if possible, or group the init values with the control sets;
- choose the initialization values to match the possible high/low constant state that may be caused by generics, logic reduction or logic removal due to optional ports usage. If a register is globally initialized to the opposite state of a unused assigned logic, the mapper may not remove the logic efficiently;
- use only positive logic for every control signal, including clock, if possible;
- have control sets grouped in number of 8 registers, whenever possible;
- do not use async clears/presets, but only synchronous sets/resets instead, if possible. This moves the clear/preset from the FFD control pins to a slice LUT, relaxing the control set constraint and increasing LUT density;
- when async reset/preset cannot be avoided, redesign the logic to have only a clear or a preset. Inference of a RS flop in XST under ISE13.1 generates very inefficient logic;

The logic in the spi_master and spi_slave cores follows these rules as much as possible, to get good LUT packing without CLB resource waste.

GOOD CLOCKING PRACTICES

Clocking in an FPGA is a very scarce resource. Although it is generally possible to access the global low-skew clock nets using signals that are generated in combinatorial paths, this almost always introduce uncertainties in clock path delays and clock phase issues between distant logic that need to meet at pipeline stages. This routing degradation can impose a lower operating speed to a design, or introduce excess jitter to state transitions, enough to cause meta stability or glitch shoot-through due to inadvertent excitation of intermediate combinatorial states. These second-order effects may lead to data setup problems at pipeline boundaries, reducing reliability of sequential circuits.

These problems affect all circuits that operate at the process limit, but clock routing integrity may worsen the problems. Therefore, good clocking practices are essential to achieve well-behaved circuitry and lower levels of headache mitigation chemical support.

There are relatively simple hard-earned rules to follow to achieve general good clocking:

- never drive a FF clock with a combinatorial LUT path. NEVER.
- do not gate a clock via a combinatorial LUT function.
- do not try to use "gates" to introduce clock delays.
- do not MUX into a clock path, unless using a global clock mux, and even then, try not to.
- for general CLB synchronous logic, use the ONE global clock for the entire circuit. Generally this will be the highest frequency clock for all the pipelines in the system. DO NOT DIVIDE IT DOWN. Don´t try to count it down and then feed lower rate circuitry with the non-clock-net signals.
- instead, use clock enables to drive lower rate sequential circuitry. You can apply the usual counters, complex combinatorial paths, MUXes and whatever you like, to drive the CE inputs of the pipelines, while the global clock spines deliver pure-blood clean clocks. This can almost always improve a circuit´s performance, and can improve dramatically the routing slack and reliability.

In this project, the SPI_MASTER core was initially designed with a low-frequency base clock input, that was to be set at 2x the SPI line bus clock. This imposed to the user the need to deliver a low frequency clock line with a clock buffer to this port. Max frequency was around 25MHz at best for the SPI clock, with reported core max at < 140MHz.

The clock regime was redesigned to have one high-speed global clock, with locally generated antiphase clock enables to drive the internal sequential logic, and the design went to 210MHz of core speed, and was tested in silicon at 50MHz of SPI clock with plenty of stability and phase accuracy.

SPEED/AREA CONSIDERATIONS

To achieve the needed performance, even the slowest FPGAs on the market can reach SPI timing closure for 20MHz using bulk LUT/FF logic for the whole model. If the intended SPI bus performance is around 15MHz~20MHz,

aggressive area optimization can be applied. In the Spartan-6 architecture, using XST and ISE13.1, the 2 blocks (master + slave) can consume 41 slices after P&R, using only LUT/FF resources.

Ultimately, the data setup times of the MISO signal seen by the sampling clock at the master input register (rx_bit_reg) will limit upper operating frequency. The SCK-to-MISO delays observed during hardware testing showed less than 2ns of delay relative to the master generated SCK line clock, using default area optimizations. This was tested at 50MHz SCK, and can probably go up to 75MHz without losing clock phase margin. Moving this register to a IOB, or applying a clock delay to the sampling clock will add slack time to the data setup, and can equalize the MISO setup to the MOSI performance, at the expense of specialized treatment, like instantiation of Xilinx-specific circuitry. The Spartan-6 IOB has a controlled signal delay element that can be used to equalize the MISO to the MOSI edges, to maximize the bus frequency. In that case, extra logic needs to be used to detect and calibrate the amount of bit phase shift needed. These techniques are not being used in this design, because the main objective here is to be as vendor-independent as possible. Such SCK delays at the master could push the SCK frequency to over 120MHz.

Another approach, if only data transmission is needed (only MOSI) from the spi_master block, is to remove the MISO-related circuitry, and operate only with the data transmission hardware. This will take the upper frequency to be limited only by the data setup needs of the receiving slave, but probably can reach 100MHz with the same area optimization.

CROSS-CLOCK DATA TRANSFER

The cores operate on two potentially asynchronous clock domains, the serial clock and the parallel interface clock.

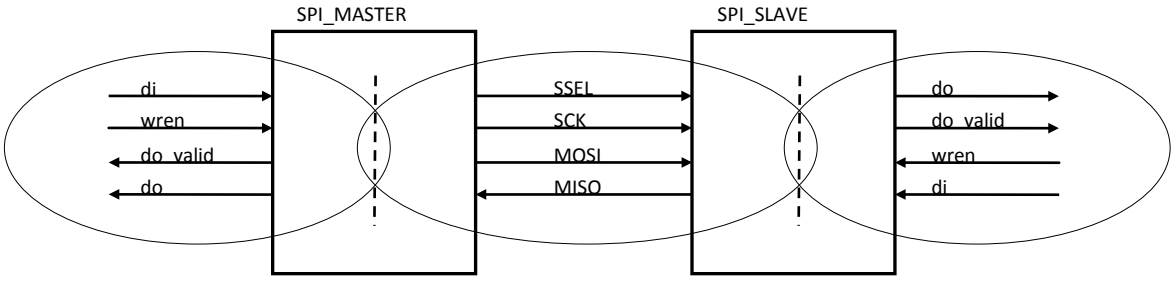


FIGURE 2 – Different clock domains for the spi cores, showing internal async bridge needs

All signals that cross clock domains are transferred by a pipelined bridge circuitry that isolate the writing and reading by the different clock domains, ensuring that all accesses have enough setup and hold margins.

PARALLEL INTERFACE TO USER CIRCUITRY

The parallel read/write interfaces have handshake lines to synchronize user circuitry to feed the SPI interface continuously, for seamless back-to-back operation. All ports in the parallel interface are synchronous to the parallel clock, 'pclk_i'.

For the write port, the user writes do port 'di_i' and strobes the signal 'wren_i'. If the SPI port is idle, i.e., if 'spi_ssel_o' = '1', the write operation starts the sending of the data written to the master write port. The signal 'di_req_o' is strobed for 2 clock cycles to prompt new data input request from the user circuitry, in time for the input buffer to be filled before the current serial data terminates transmission. The generic parameter 'PREFETCH' controls how many 'spi_sck_o' clock cycles in advance to pre-fetch parallel data. Depending on the user circuitry that is feeding the data, more bus cycles are needed to produce the required data. This can be adjusted by setting 'PREFETCH' at instantiation to the needed cycles. If the pre-fetch request is ignored by the master, and no 'wren_i' strobe is sent before the last serial bit is transferred, the transmission is stopped and 'spi_ssel_o' returns to the idle state '1'. For the slave operation, the 'di_req_o' prefetch signal prompts the user to present the next data word to be read by the line. If the prefetch is ignored, the data at the parallel input port is read anyway.

For the read port, the output signal 'do_o' is the registered output of the received data. The receive register is updated with the received data at the end of the last bit received. The signal 'do_valid_o' is then strobed for 2 'pclk_i' clock cycles, after 3 cycles have elapsed, to guarantee no data setup glitches on the data transfer. So, data will be available for reading 4 'pclk_i' cycles after the last bit of the received word, and will remain valid for the whole next SPI cycle, until a new word is received. If the transmission is interrupted before the last bit by a 'rst_i' interface reset or a SSEL deselect, the parallel output buffer holds the last fully received word. When transmission completes and the interface goes to idle state ('spi_ssel_o' = '1'), the output port holds the last received word. See the scope screenshots below for the master/slave interaction for a 3-word transfer on a 8-bit configuration.

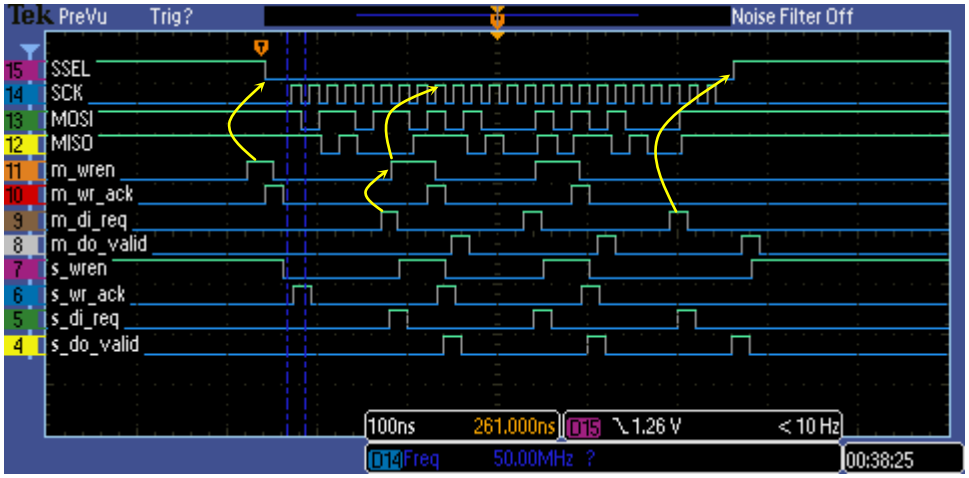


Figure 3 – Continuous transfer using the PREFETCH signals

The picture shows a sequence of events:

1- the transmission starts when the master input parallel port is written and wren is strobed.

2- The di_req signal requests the next word, 3 SCK cycles before word's end (PREFETCH = 3).

3- The master di_i port is written and wren is strobed, before last bit. This reloads the shift register and transmission of the next word starts.

4- At the last word, the prefetch signal is ignored by the user circuit, and the transmission stops after the last data bit.

At the slave interface, the prefetch allows the user to update the parallel data in time to be consumed by the stream.

3

Architecture

Each core is implemented as a single design entity. The block diagram for each core is detailed below:

Each core has 2 interfaces, the SPI bus and the parallel data I/O ports. Separate clock domains inside the cores synchronize the operations of the core RTL registers and the parallel I/O ports.

Small but significant differences exist in the state machines of the master and slave functions to have specialized cores for each function. Instead of making a universal master/slave core with runtime selection of operation mode, the function and mode are selected during instantiation, using generics, to achieve efficient silicon usage.

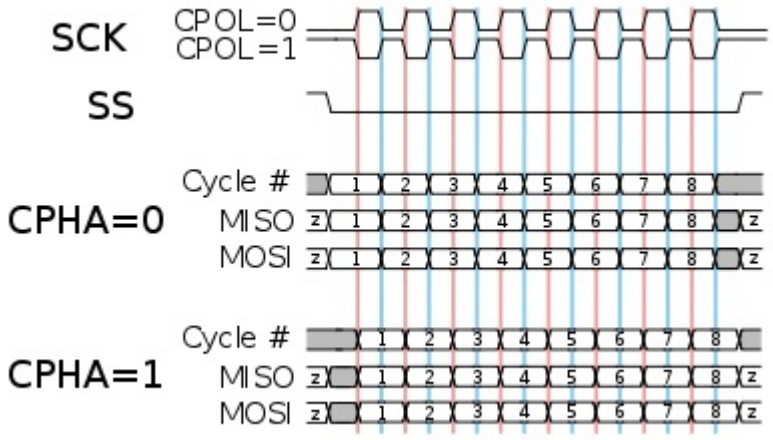
The data transfer between the clock domains is handled by async bridge pipelines, that transfer the synchronization signals from the write port to the core and from the core to the read port. The parallel data ports are read and written to by the core with plenty of data setup and hold time, and the user synchronization signals control user access to the ports, to avoid setup sampling problems.

4

Operation

The internal logic of each core is a sequencer implemented as a single RTL state machine. The state machine is clocked by the SPI SCK clock. The spi_master block generates the spi clock from a 2x input clock, using 2 FFDs to derive two in-phase clocks, one continuous clock to control the sequencer, and an output spi clock, that is controlled with the CE input of a second FFD. Both clocks have high phase correlation, so serial data change is synchronous to the output SCK generated.

The SPI bus has 4 modes of operation, controlled by 2 parameters: Clock Polarity (CPOL) and Clock Phase (CPHA). The master and slave in a SPI connection must have the same SPI mode to interoperate. The modes are depicted in the following waveform diagram.



Serial data output signal changes at the clock edge selected by CPOL and CPHA.

The serial data input is sampled at the opposite clock edge. Data setup time to the data sampling edge is the limiting factor for maximum SPI operating frequency. If transmit-only operation is intended, the master can achieve a much higher clock frequency.

The model has generics to control generation of SPI mode, word width and data prefetch timing.

The operation of the spi_master block starts with a write to the parallel data in port.

5

Clocks

[This section specifies all the clocks. All clocks, clock domain passes and the clock relations should be described.]

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
clk_pad_i	Input Pad	10	4	0.1	Duty cycle 70/30.	For external interface.
wb_clk_I	PLL	200	-	-	Must be synchronized to sm_clk_i	System clock.
sm_clk_i	Input port	55	40	1	There are multi-clocks paths.	Clock 55MHz for State machine.

Table 1: List of clocks

6

IO Ports

[This section specifies the core IO ports.]

Port	Width	Direction	Description
wb_clk_i	1	Input	Block's WISHBONE Clock Input
wb_rst_i	1	Input	Block's WISHBONE Reset Input
wb_sel_i	4	Input	Block's WISHBONE Select Inputs
foo_pad_o	1	Output	Block's foo output to output pad
…			

Table 2: List of IO ports

Appendix A

Name

[This section may be added to outline different specifications.]

Appendix B

Name

[This section may be added to outline different specifications.]

Index

[This section contains an alphabetical list of helpful document entries with their corresponding page numbers.]