Generic Soft SPI Master Controller

Reference Design

FPGA-RD-02209-1.0

December 2020

Disclaimers

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
FPGA	Field-Programmable Gate Array
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
SCLK	Serial Clock Signal
SPI	Serial Peripheral Interface
SS	Slave Select

1. Introduction

The Serial Peripheral Interface (SPI) bus provides an industry standard interface between processors and other devices. This reference design documents an SPI Master Controller designed to provide an interface between a generic processor with parallel bus interface and external SPI devices.

SPI interface is a good choice for designs that require full-duplex capability for sending and receiving data at the same time. The SPI Master Controller can communicate with multiple off-chip SPI ports and can also be configured to support all modes of CPOL and CPHA (00, 01, 10, and 11).

This reference design implements a Soft SPI Master Controller Module on any Lattice FPGA using Lattice Diamond® 3.11 and Lattice Radiant™ 2.1.

2. Features

- Supports a wide array of Lattice FPGAs such as MachXO2™, MachXO3™, LatticeECP3™, ECP5™, CrossLink™, CrossLink™-NX, and iCE40 UltraPlus™
- Provision for easy integration of any processor interface
- Up to five slave select outputs
- Compatible with all SPI Modes
- Configurable timing features for timing-sensitive slave devices
    - Configurable SCLK frequency
    - Configurable interval between assertion of SS_N and the first SCLK clock edge
    - Configurable interval between the last SCLK clock edge and deassertion of SS_N
    - Configurable interval between SPI data bytes

3. Functional Description

3.1. Block Diagram

Figure 3.1 shows an overview of the reference design with two interfaces: the Processor Interface (left side arrows) and the SPI Interface (right side arrows). The Processor Interface can be connected internally on the same FPGA device or externally to an external application processor. The SPI Interface may be connected to a maximum of five slave devices. The code, however, can be modified if more slave devices are required.

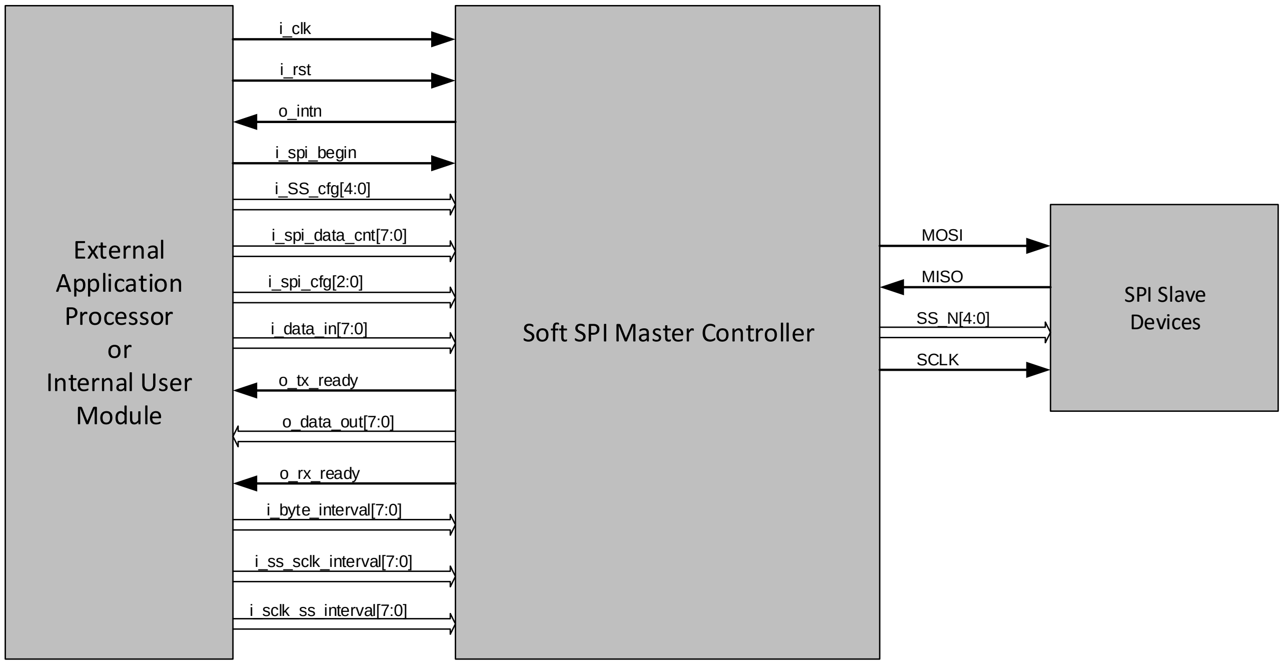


Figure 3.1. Block Diagram

3.2. Signal Descriptions

Table 3.1. Signal Descriptions

Signal	Width	Type	Description
i_clk	1	Input	System clock
i_rst	1	Input	Asynchronous active high reset
o_intn	1	Output	Active low interrupt Indicates that the SPI transaction has completed
i_spi_begin	1	Input	Asserted momentarily to begin SPI transaction Active high
i_SS_cfg	5	Input	Defines which of the five slave select pins are activated during a SPI transaction Active high.
i_spi_data_cnt	8(default)	Input	Sets the number of bytes in the SPI transaction The width can be increased by modifying the DATA_CNT_WIDTH parameter. Refer to Table 5.1 for more information.
i_spi_cfg	3	Input	Defines the Direction, Clock Phase (CPHA), and Clock Polarity (CPOL) of each SPI transaction. • i_spi_cfg[2]: (Direction) • 0 = MSB First • 1 = LSB First • i_spi_cfg[1]: (Clock Phase) • 0 = data should be captured at the leading edge. • CPHA = 0 • 1 = data should be captured at the trailing edge. • CPHA = 1 • i_spi_cfg[0]: (Clock Polarity) • 0 = SCLK idles at 0 • CPOL = 0 • 1 = SCLK idles at 1 • CPOL = 1
i_data_in	8	Input	Parallel input data from the processor interface Captured data is sent to the MOSI line.
o_tx_ready	1	Output	Positive strobe to indicate that data is captured from the *i_data_in* port.
o_data_out	8	Output	Parallel output data sent to the processor interface Data from this port comes from the MISO line.
o_rx_ready	1	Output	Positive strobe to indicate that data can be read from the *o_data_out* port.
i_byte_interval	8	Input	Defines the interval between SPI data bytes in terms of i_clk cycles. • Allowable value is from 1 to 255.
i_ss_sclk_interval	8	Input	Defines the interval between the SS_N assertion to low and the first SCLK edge. • Allowable value is from 1 to 255.
i_sclk_ss_interval	8	Input	Defines the interval between the SS_N deassertion to high and the last SCLK edge. • Allowable value is from 1 to 255.
MISO	1	Input	SPI data bus – master in, slave out
MOSI	1	Output	SPI data bus – master out, slave in
SS_N	5	Output	SPI slave select outputs Active low
SCLK	1	Output	SPI serial clock

3.3. Processor Interface Timing Diagram

The following describes the timing for the Processor Interface as illustrated by Figure 3.2. You only need to control and interpret these ports and they are automatically translated to the SPI Interface ports. For simplicity, the SPI Interface ports are not shown here. To see how the Processor Interface and SPI Interface ports align, refer to the HDL Simulation and Verification section of this document.

1. The Reference Design starts in an idle state. The Internal User Module or External Application Processor can prepare the inputs for the *i_SS_cfg*, *i_spi_data_cnt* and *i_spi_cfg* ports as described in Table 3.1. Afterwards, the *i_spi_begin* port needs to be asserted for 1 clock cycle to begin the SPI transaction with these defined settings.
2. After a few clock cycles, a positive *o_tx_ready* strobe is generated to signify that input data for the *i_data_in* port should be ready. During this point, each bit of data is captured and subsequently sent to the MOSI SPI port. The *i_data_in* data value should be held until the next *o_tx_ready* strobe.
3. After a few clock cycles, a positive *o_rx_ready* strobe is generated to signify that output data from the *o_data_out port* is ready and can be utilized by the Internal Module or External Application Processor. The data captured from this port is the data received from the MISO SPI port.
4. When more than one byte of data is defined in the *i_spi_data_cnt* port, steps 2 and 3 are automatically repeated until the defined number of bytes is reached.
5. When the total number of bytes defined in the *i_spi_data_cnt* input is reached, an *o_intn* interrupt is generated and the reference design returns to an idle state.

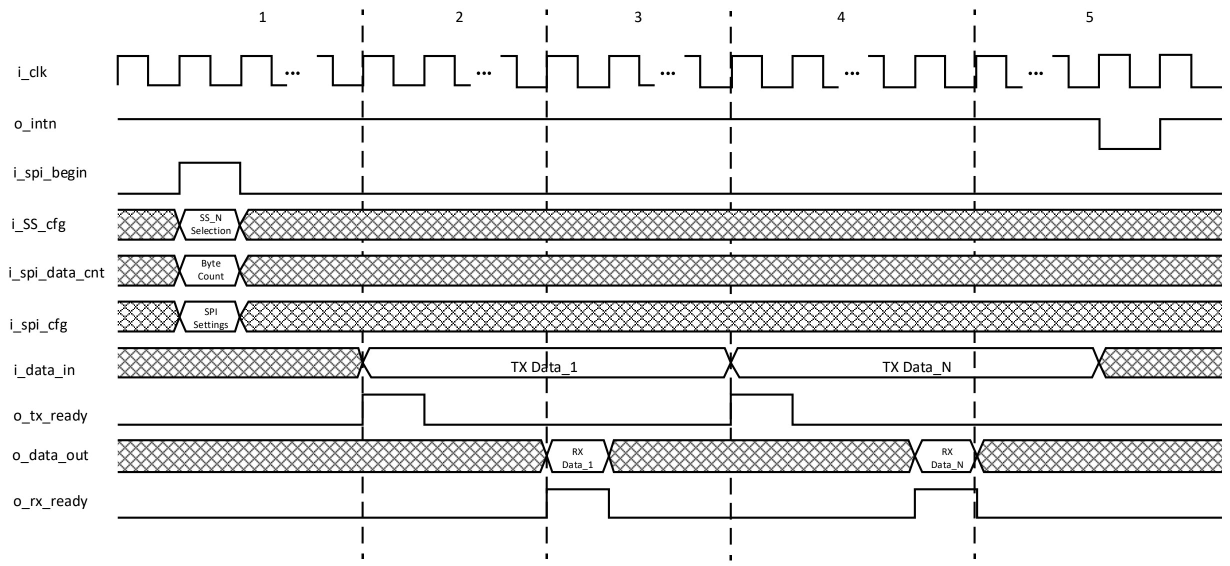


Figure 3.2. Processor Interface Timing Diagram

3.4. SPI Interface Timing Diagram

Figure 3.3 shows the timing diagram of all SPI Modes (CPOL and CPHA combinations) with the direction set to *MSB First* (refer to Table 3.1). When the Processor Interface ports are properly controlled, the SPI interface drives the MOSI line and samples the MISO line based on the CPOL/CPHA modes as follows:

- At CPOL=0, the base value of the clock is zero
  - For CPHA=0, data is read on the clock's rising edge and the data is changed on the falling edge (SPI Mode 0).
  - For CPHA=1, data is read on the clock's falling edge and the data is changed on the rising edge (SPI Mode 1).
- At CPOL=1, the base value of the clock is one (inversion of CPOL=0)
  - For CPHA=0, data is read on the clock's falling edge and the data is changed on the rising edge (SPI Mode 2).
  - For CPHA=1, data is read on the clock's rising edge and the data is changed on the falling edge (SPI Mode 3).

This reference design pushes bits of data on the MOSI line from a shift register based on bit count (*data_cnt*) and CPOL/CPHA modes. It also samples the MISO line and shifts the data based on the current bit count of the SPI transaction as well as the CPOL and CPHA modes.

During each SPI clock cycle (*SCLK*), a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line.
- The slave sends a bit on the MISO line; the master reads it from that same line.

While all four of these operations happen each cycle, they may not be used or required. It is up to the designer to set the proper command and data bytes framing to make it meaningful to a particular application.

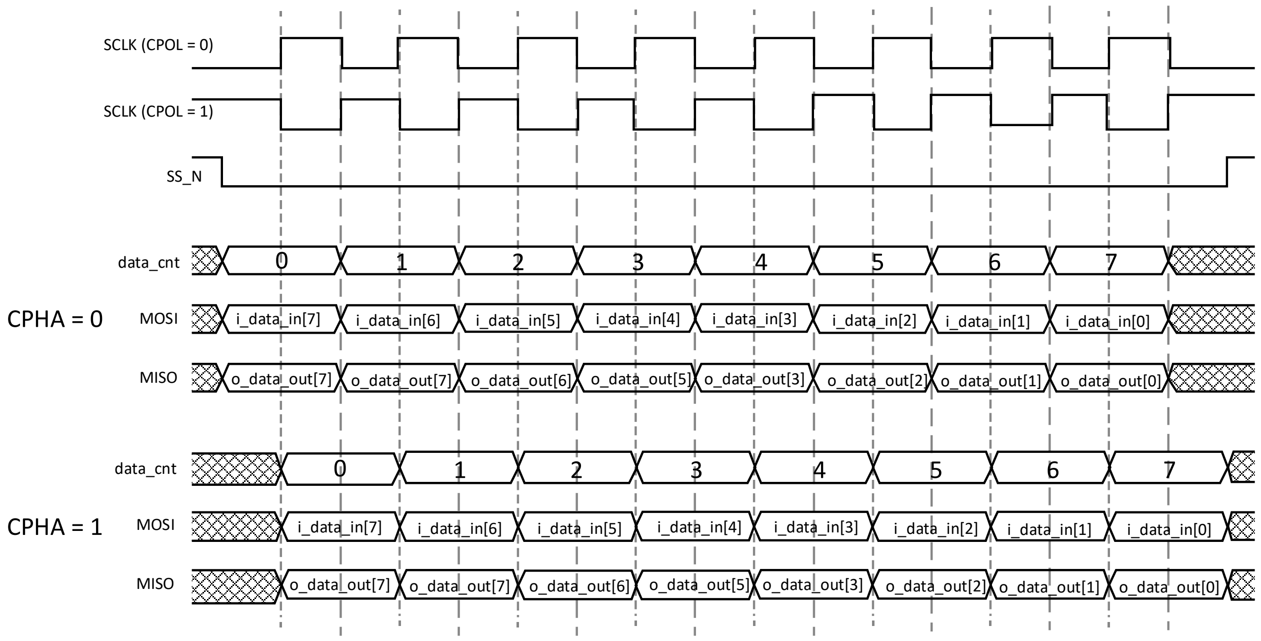


Figure 3.3. SPI Interface Timing Diagram (MSB First)

4. Operation Sequence

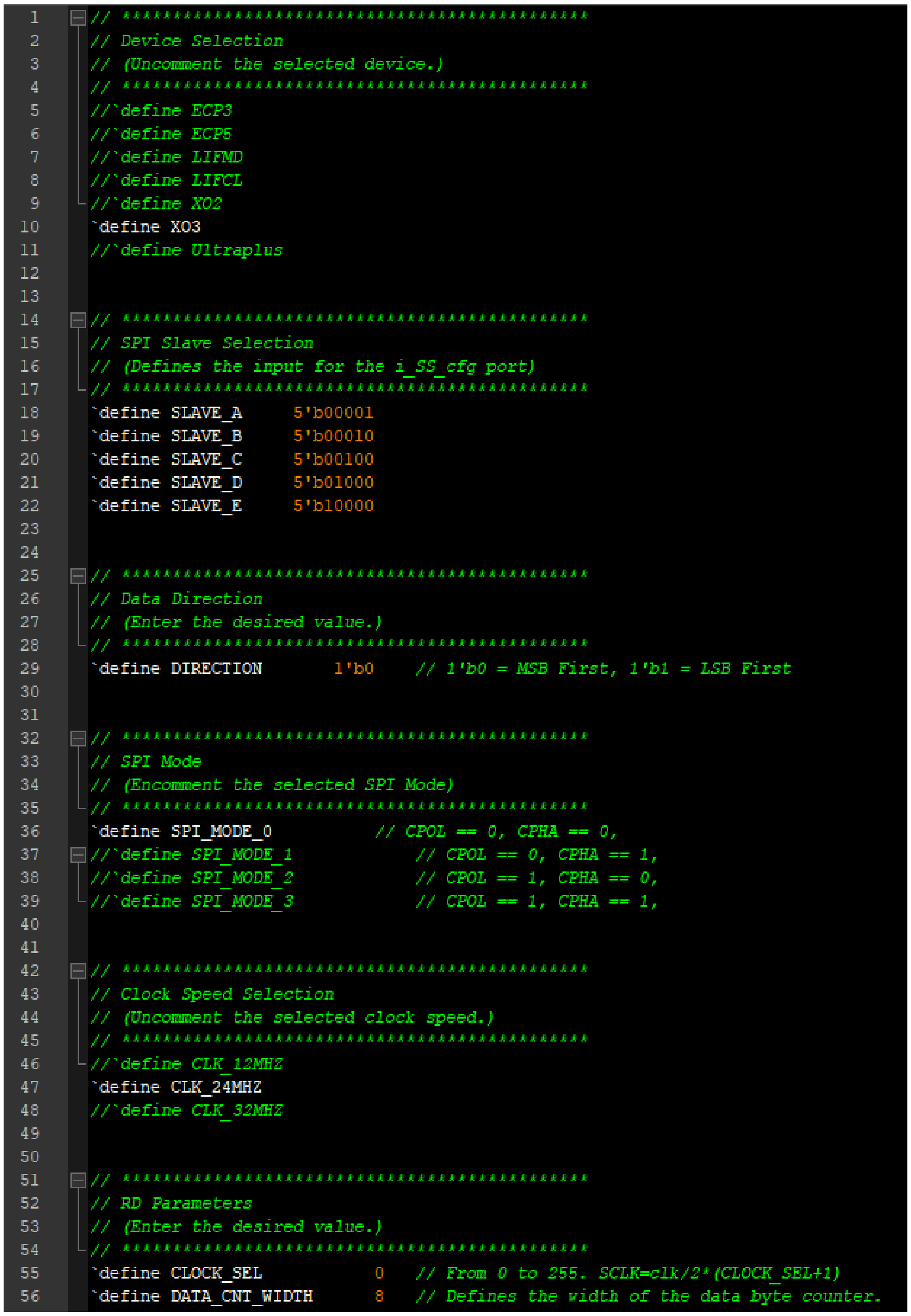
Command and data byte framing varies across different manufacturers of slave devices. You should refer to the manufacturer's datasheet of the selected slave device and implement this reference design based on the specific requirements mentioned. As an example, a companion demo was created showing actual SPI transactions to an external SPI Flash device with proper command and data bytes framing. Refer to the Generic Soft SPI Master Controller Demo (FPGA-UG-02123).

5. Customization

To customize the testbench files of this reference design, a file named *tb_defines.v* contains all the compiler directives that you can modify. This includes SPI Slave Selection, Data Direction, SPI Mode, Clock Speed, and others. Table 5.1 shows the complete list of compiler directives. Figure 5.1 shows an example of customization implemented in the *tb_defines.v* file.

Table 5.1. Compiler Directives Options

Category	Compiler Directives	Remarks
Device Selection	ECP3	Uncomment only one to enable the selected device.
	ECP5	
	LIFMD	
	LIFCL	
	MachXO2	
	MachXO3	
	iCE40 UltraPlus	
SPI Slave Selection	SLAVE_A	Defines the input for the *i_SS_cfg* port.
	SLAVE_B	
	SLAVE_C	
	SLAVE_D	
	SLAVE_E	
Data Direction	DIRECTION	1'b0 = MSB First, 1'b1 = LSB First
SPI Mode	SPI_MODE_0	Uncomment only one to enable a defined CPHA and CPOL combination.
	SPI_MODE_1	
	SPI_MODE_2	
	SPI_MODE_3	
Clock Speed Selection	CLK_12MHZ	Uncomment only one to enable the selected clock speed.
	CLK_24MHZ	
	CLK_32MHZ	
RD Parameters	CLOCK_SEL	Defines the SCLK frequency based on the clock source frequency with the following formula: *SCLK=clk/2*(CLOCK_SEL+1)* Allowable value is from 0 to 255.
	DATA_CNT_WIDTH	Defines the width of the data byte counter. Increasing this value increases the maximum number of data bytes in each SPI transaction.



```
// **********************************************
// Device Selection
// (Uncomment the selected device.)
// **********************************************
//`define ECP3
//`define ECP5
//`define LIFMD
//`define LIFCL
//`define XO2
`define XO3
//`define Ultraplus


// **********************************************
// SPI Slave Selection
// (Defines the input for the i_SS_cfg port)
// **********************************************
`define SLAVE_A     5'b00001
`define SLAVE_B     5'b00010
`define SLAVE_C     5'b00100
`define SLAVE_D     5'b01000
`define SLAVE_E     5'b10000


// **********************************************
// Data Direction
// (Enter the desired value.)
// **********************************************
`define DIRECTION       1'b0    // 1'b0 = MSB First, 1'b1 = LSB First


// **********************************************
// SPI Mode
// (Encomment the selected SPI Mode)
// **********************************************
`define SPI_MODE_0          // CPOL == 0, CPHA == 0,
//`define SPI_MODE_1            // CPOL == 0, CPHA == 1,
//`define SPI_MODE_2            // CPOL == 1, CPHA == 0,
//`define SPI_MODE_3            // CPOL == 1, CPHA == 1,


// **********************************************
// Clock Speed Selection
// (Uncomment the selected clock speed.)
// **********************************************
//`define CLK_12MHZ
`define CLK_24MHZ
//`define CLK_32MHZ


// **********************************************
// RD Parameters
// (Enter the desired value.)
// **********************************************
`define CLOCK_SEL          0   // From 0 to 255. SCLK=clk/2*(CLOCK_SEL+1)
`define DATA_CNT_WIDTH     8   // Defines the width of the data byte counter.
```

Figure 5.1. Compiler Directive Customization Example

6. HDL Simulation and Verification

This Generic Soft SPI Master Controller reference design is simulated using a top-level testbench file *tb.v* that acts as the application processor mentioned in Figure 3.1. There is also a separate testbench file *spi_slave.v* with two instantiations, which the SPI Master is able to communicate with at certain points in the simulation.

The simulation shown Figure 6.1 and Figure 6.2 runs in SPI Mode 0 (CPHA = 0, CPOL = 0) with the direction of the data sending the MSB first *(i_spi_cfg[2] = 0).* For simplicity, only selected signals are shown in the figures below. The following lists the testbench flow:

1. As shown in Figure 6.1, the SPI Master Controller performs a 5-byte transaction to the first SPI Slave.
    a. The SPI Master sends 0x00, 0x11, 0x22, 0x33, and 0x44 to the MOSI port. Note that these are the data captured by the SPI Master Controller from the *i_data_in* input beginning at each momentary assertion of the *o_tx_ready* output.
    b. The SPI Master receives 0xEE, 0xDD, 0xCC, 0xBB, and 0xAA from the MISO port. Note that these are the data that can be captured by the Application Processor from the *o_data_out* output during momentary assertion of the *o_rx_ready* output.
    c. The Slave Select (SS) input of the SPI Slave is connected to the *SS_N[0]* port of the SPI Master.
2. As shown in Figure 6.2, the SPI Master performs a 2-byte transaction to the second SPI Slave with some timing adjustments applied to the *i_byte_interval, i_ss_sclk_interval,* and *i_sclk_ss_interval* inputs of the SPI Master Controller*.*
    a. The SPI Master sends 0x00 and 0x11 to the MOSI port. Note that there is now a larger interval between the falling edge of *SS_N[1]* and the first edge of the *SCLK* port.
    b. The SPI Master receives 0xEE and 0xDD from the MISO port. Note that there is now a larger interval between the SCLK edges of the two data bytes.
    c. The Slave Select (SS) input of the SPI Slave is connected to the *SS_N[1]* port of the SPI Master.
3. For easier analysis, the top-level testbench file *tb.v* implements display tasks ($display) showing the simulation activity and in what timeline a certain task is performed as shown in Figure 6.3.

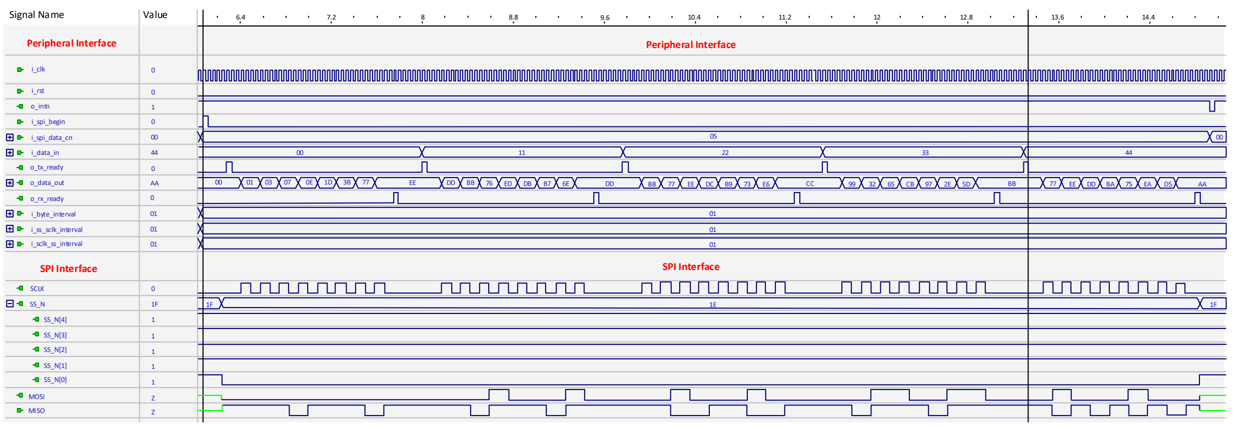


Figure 6.1. 5-Byte Transaction to the First SPI Slave

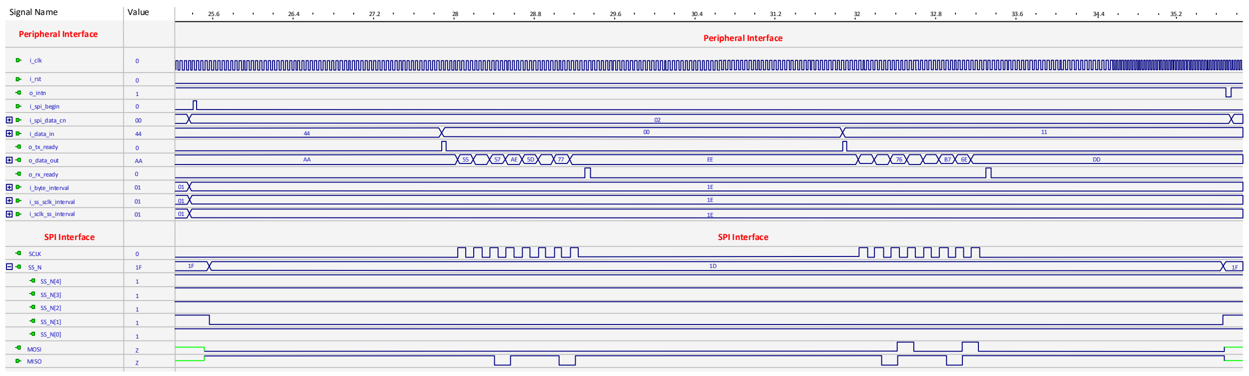
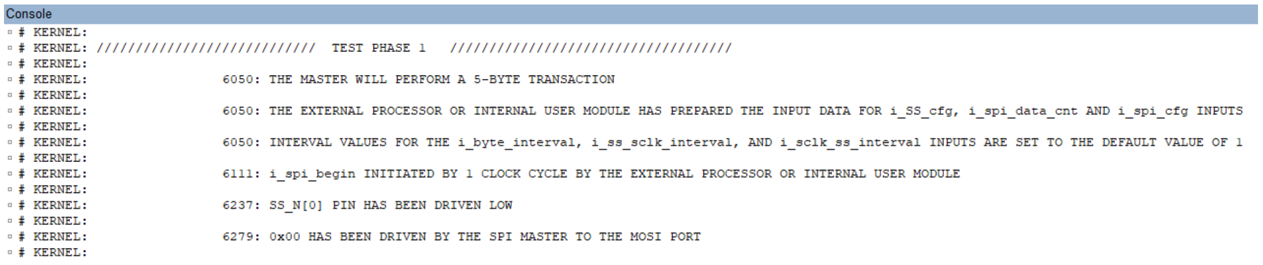


Figure 6.2. 2-Byte Transaction to the Second SPI Slave



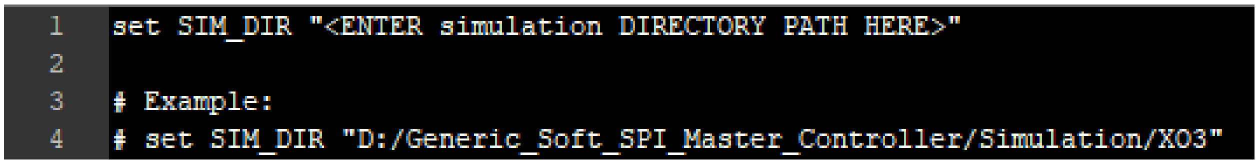
```
Console
# KERNEL:
# KERNEL: ////////////////////////////  TEST PHASE 1   ////////////////////////////////////
# KERNEL:
# KERNEL:                  6050: THE MASTER WILL PERFORM A 5-BYTE TRANSACTION
# KERNEL:
# KERNEL:                  6050: THE EXTERNAL PROCESSOR OR INTERNAL USER MODULE HAS PREPARED THE INPUT DATA FOR i_SS_cfg, i_spi_data_cnt AND i_spi_cfg INPUTS
# KERNEL:
# KERNEL:                  6050: INTERVAL VALUES FOR THE i_byte_interval, i_ss_sclk_interval, AND i_sclk_ss_interval INPUTS ARE SET TO THE DEFAULT VALUE OF 1
# KERNEL:
# KERNEL:                  6111: i_spi_begin INITIATED BY 1 CLOCK CYCLE BY THE EXTERNAL PROCESSOR OR INTERNAL USER MODULE
# KERNEL:
# KERNEL:                  6237: SS_N[0] PIN HAS BEEN DRIVEN LOW
# KERNEL:
# KERNEL:                  6279: 0x00 HAS BEEN DRIVEN BY THE SPI MASTER TO THE MOSI PORT
# KERNEL:
```

Figure 6.3. Aldec Active-HDL Console View

6.1. Using the Simulation File (.DO)

To use the simulation file, perform the following steps:

1. Open the DO file on a text editor and replace the text **<ENTER simulation DIRECTORY PATH HERE>** from Line 1 with the directory path of the simulation file. An example is seen on Line 4 of the file.



```
1   set SIM_DIR "<ENTER simulation DIRECTORY PATH HERE>"
2
3   # Example:
4   # set SIM_DIR "D:/Generic_Soft_SPI_Master_Controller/Simulation/XO3"
```

Figure 6.4. Changing the Simulation Directory

2. Run the file on Aldec Active-HDL by selecting *Execute macro…* under the **Tools** option.

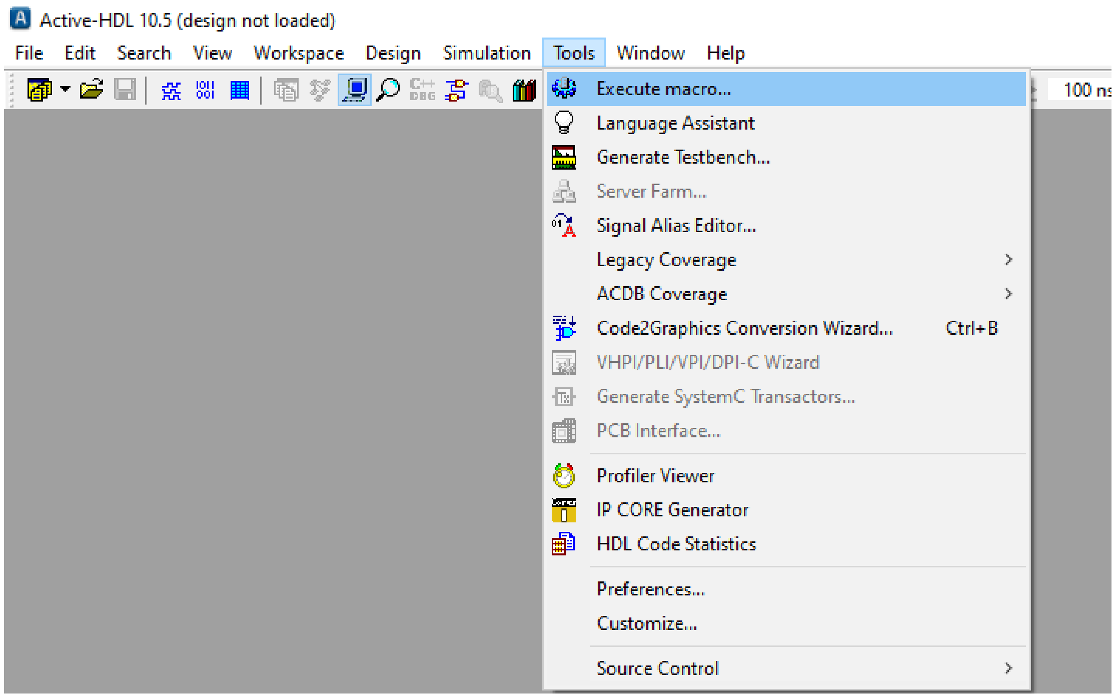


Figure 6.5. Running the Simulation File

7. Packaged Design

The reference design folder (Generic_Soft_SPI_Master_Controller) contains five subfolders: Docs, Project, Simulation, Source, and Testbench. The details of each subfolder are as follows:

- Project – contains subfolders for each FPGA Family. Each of these subfolders contains either a Diamond or a Radiant project file (.LDF and .RDF).
- Simulation – contains subfolders for each FPGA Family. Each of these subfolders contains the simulation file (.DO) used to run RTL simulation on Aldec Active-HDL.
- Source – contains the main source code file named *spi_master_controller.*v.
- Testbench – contains all the testbench source files named *tb.v, tb_defines.v,* and *spi_slave.v*.

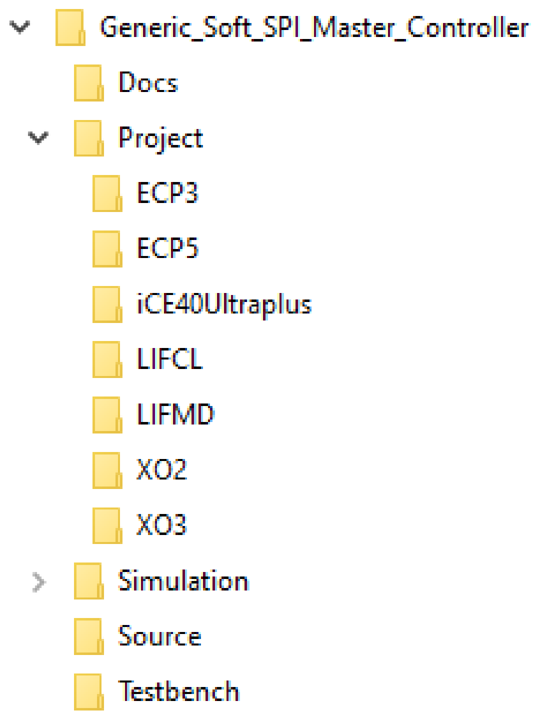


Figure 7.1. Packaged Design Directory Structure

8. Hardware Validation

This reference design was hardware validated using a MachXO3- 9400 Development Board (LCMXO3LF-9400C-ASC-BEVN). A companion demo was also created to allow you to perform actual hardware validation on most Lattice FPGA. Refer to the Generic Soft SPI Master Controller Demo (FPGA-UG-02123).

9. Implementation

This design is implemented in Verilog. When using this design in a different device or strategy settings, density, speed/grade, performance, and utilization may vary. Due to the limitations of the I/O pin count of iCE40 UltraPlus and CrossLink devices, the included two projects for these fail during Map. However, if most of the ports for this reference design are only used internally, Map succeeds like in the case of the companion demo, Generic Soft SPI Master Controller Demo (FPGA-UG-02123).

Table 9.1. Resource Utilization

Device Family	Language	Utilization (LUTs)	f_{MAX} (MHz)	I/O
Lattice ECP3[1]	Verilog	205	90	This Reference Design has a total of 70 ports. The hardware validated companion demo mentioned in this document is only using 15 I/O since most of the ports are only used internally.
ECP5[2]	Verilog	220	71	
CrossLink[3]	Verilog	~220[8]	49	
CrossLink-NX[4]	Verilog	218	75	
iCE40 UltraPlus[5]	Verilog	211[9]	28	
MachXO2[6]	Verilog	213	72	
MachXO3[7]	Verilog	213	72	

Notes:

1. Performance and utilization characteristics are generated using LFE3-35EA-8FN484C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro®.
2. Performance and utilization characteristics are generated using LFE5U-85F-8BG381C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
3. Performance and utilization characteristics are generated using LIF-MD6000-6MG81I with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
4. Performance and utilization characteristics are generated using LIFCL-40-7BG400I with Lattice Radiant 2.0 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
5. Performance and utilization characteristics are generated using iCE40UP5K-SG48I with Lattice Radiant 2.0 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
6. Performance and utilization characteristics are generated using LCMXO2-7000HE-6TG144C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
7. Performance and utilization characteristics are generated using LCMXO3LF-9400C-6BG484C with Lattice Diamond 3.11 design software with either LSE (Lattice Synthesis Engine) or Synplify Pro.
8. Approximation only. The Selected CrossLink device does not meet the required 70 I/O for this reference design. However, if some of the ports are going to be utilized internally, this reference design can still be used.
9. Total LUT count came from the Map Resource Usage section of Lattice Radiant software's report browser after compiling the design using another top-level unit of the companion demo that instantiates the *spi_master_controller* module.

References

For more information, refer to the following documents:

- LatticeECP3 EA Family Data Sheet (DS1021)
- ECP5 and ECP5-5G Family Data Sheet (FPGA-DS-02012)
- CrossLink Family Data Sheet (FPGA-DS-02007)
- MachXO2 Family Data Sheet (DS1035)
- MachXO3 Family Data Sheet (FPGA-DS-02032)
- iCE40 UltraPlus Family Data Sheet (FPGA-DS-02008)
- CrossLink-NX Family Data Sheet (FPGA-DS-02049)
- Generic Soft SPI Master Controller Demo (FPGA-UG-02123)

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, December 2020

Section	Change Summary
All	Initial release.

www.latticesemi.com