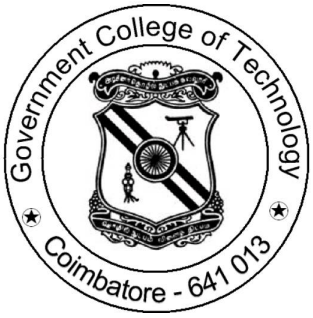



# SPI IP DESIGN USING VERILOG

MINI PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF **BACHELOR OF ENGINEERING** IN **ELECTRONICS AND COMMUNICATION ENGINEERING** OF THE ANNA UNIVERSITY

**MINI PROJECT WORK**

**2022**

Submitted by

**ANTO SANJAY S**

**1914104**

**GOWTHAM S**

**1914113**

**KAVINRAJ K**

**1914122**

**TAMILARASAN M**

**1914L14**

**Under the Guidance of**

**Dr. O. SARANIYA**

**Professor (CAS).,ECE**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**GOVERNMENT COLLEGE OF TECHNOLOGY**

(An Autonomous Institution affiliated to Anna University)

**COIMBATORE - 641 013**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

# GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University)

**COIMBATORE - 641 013**

## MINI PROJECT WORK

**DECEMBER 2022**

This is to certify that this mini project work entitled

## SPI IP DESIGN USING VERILOG

is the bonafide record of mini project work done by


**ANTO SANJAY S**

**1914104**

**GOWTHAM S**

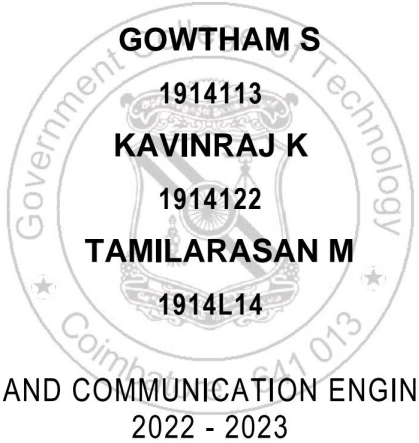
**1914113**

**KAVINRAJ K**

**1914122**

**TAMILARASAN M**

**1914L14**

of B.E. (ELECTRONICS AND COMMUNICATION ENGINEERING) during the year 2022 - 2023

| | |
|---|---|
| Project Guide | Head of the Department |
| **Dr. O. SARANIYA, M.E., Ph.D.,** | **Dr. C. SANTHI, M.E., Ph.D.,** |

Submitted for the mini project viva-voce examination held on _____

| | |
|---|---|
| Internal Examiner | External Examiner |

# ACKNOWLEDGEMENT

# ACKNOWLEDGEMENT

We oblige ourselves to the Almighty who has given us all the strengths and knowledge to complete this Final year Mini Project work.

We wish to thank **Dr. P. THAMARAI**, Principal, Government College of Technology, Coimbatore for providing us the necessary facilities for completing the Mini Project work.

We express our heartiest thanks to **Dr. C. SANTHI**, Head of the Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore for providing us the guidelines which are helpful in the Mini Project work.

We wish to enunciate our deep sense of gratitude and loyalty to our Project Guide **Dr. O. SARANIYA**, Professor (CAS) in Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore for being the beacon of guidance of our Mini Project work and motivating us in every possible manner, which led to the successful completion of the project work.

We wish to articulate our thanks to our project committee members **Dr. P. DEEPA**, Assistant Professor in Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore and **Dr. G. THIRUGNANAM**, Associate Professor (CAS) in Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore for their ideas in improving this project work.

We also thank the technical and non-technical staff in the Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore for their support.

Finally, we are also grateful to our parents and friends for their motivation, blessings, affection, and their loving co-operation from the day of this academic venture.

# ABSTRACT

# ABSTRACT

Communication has always been one of the top most concerns for human civilization. Ever since speech has originated millennia ago, Communication has evolved in a variety of ways from symbols to cave paintings. With the advent of modern technology, the range and efficiency of communication has increased many folds. Nonetheless, new methods and ways were required to improve the range of communication and retain the accuracy of the information. Then came numerous protocols into existence to meet the demands like I2C, Zigbee, UART, SPI etc. The objective of this Project is to design and implement the SPI communication protocol module using FPGA design flow in Verilog HDL. The Serial Peripheral Interface module allows synchronous, full duplex serial communication between the microcontroller unit and peripheral devices. The module was designed and simulated using Verilog HDL in QUARTUS II 10.0 web edition Software Tool design Suite.

# CONTENTS

# CONTENTS

# LIST OF FIGURES

## LIST OF FIGURES

# LIST OF TABLES

## LIST OF TABLES

# INTRODUCTION

# CHAPTER 1

# INTRODUCTION

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid-1980s and has become a de facto standard. Typical applications include Secure Digital cards and liquid crystal displays.

SPI devices communicate in full duplex mode using a master-slave architecture usually with a single master (though some Atmel and Silabs devices support changing roles on the fly depending on an external (SS) pin). The master (controller) device originates the frame for reading and writing. Multiple slave-devices may be supported through selection with individual chip select (CS), sometimes called slave select (SS) lines.

Sometimes SPI is called a four-wire serial bus, contrasting with three-, two-, and one-wire serial buses. The SPI may be accurately described as a synchronous serial interface, but it is different from the Synchronous Serial Interface (SSI) protocol, which is also a four-wire synchronous serial communication protocol. The SSI protocol employs differential signalling and provides only a single simplex communication channel. For any given transaction SPI is one master and multi slave communication.

# LITERATURE SURVEY

# CHAPTER 2

# LITERATURE SURVEY

[1] Karamalaputti, P., Kumar, A, 2022, “Design and Simulation of Serial Peripheral Interface Protocol Using Pulsed Latches”. First International Conference on Computational Electronics for Wireless Communications. Lecture Notes in Networks and Systems, Springer, Singapore, vol 329.

In this paper, Communication protocols are useful for transfer of data between devices. Even though we have many protocols, SPI is one of the most important bus protocols. SPI is used to connect microprocessor to the peripheral devices. It works on the principle of FIFO (Ring topology). Normally, Flipflops were used to synchronize the data between the Master and Slave but it consumes more power. In this paper, SPI is implemented using Pulsed Latches to reduce the power consumption. Pulsed latches consume less power, and area occupied by the pulsed latches is less compared to Flipflops. The design is simulated using Verilog. The Integrated Development Environment (IDE) used is Xilinx-ISE, and ISE Simulator (ISIM) is used to verify the waveforms.

[2] Orhan Gazi & A. Çagri Ari, 2021, “Serial Peripheral Interface. In: State Machines using VHDL” Springer, Cham.

Serial communication can be divided into two main categories, one is the asynchronous communication and the other is the synchronous communication. RS232 communication is an asynchronous communication type in which start and stop bits are used for the controlling of communication, and the use of controlling bits reduces the transmission efficiency. In synchronous serial communication both the transmitter and receiver use the same clock signal. For this reason, in synchronous communication at least two wires, one is used for clock and the other one is used for data, should be available between transmitter and receiver modules. Serial peripheral

interface (SPI) is a synchronous communication protocol developed by Motorola company for the synchronous serial communication of 68HC family of microcontrollers by its peripherals. Later, this standard became de-facto for serial communication, and it has been adapted by many companies for their products. In this chapter, we first give information about synchronous serial communication and SPI protocol, and then explain how to write VHDL codes to implement SPI protocol for FPGA devices so that they can communicate with electronic devices utilizing SPI communication ports.

[3] Dawoud Shenouda Dawoud, Peter Dawoud, 2020, “Serial Peripheral Interface (SPI),” IEEE, River Publishers, 191 – 244.

Data communication standards are comprised of two components: The "protocol" and "Signal/data/port specifications for the devices involved". The protocol describes the format of the message and the meaning of each part of the message. To connect any device to the bus, an external device must be used as an interface which will put the message in a form which fulfils all the electrical specifications of the port. These specifications are called the "Standard". The most famous such serial communication standard is the RS-232. In IT technology, Communication can be serial or parallel. Serial communication is used for transmitting data over long distances. It is much cheaper to run the single core cable needed for serial communication over a long distance than the multicore cables that would be needed for parallel communication. It is the same in wireless communication: Serial communication needs one channel while parallel needs multichannel. Serial Communication can also be classified in many other ways, for example synchronous and asynchronous; it can also be classified as simplex, duplex and half duplex. Because of the wide spread of serial communication from home automation to sensor and controller networks, there is a need for a very large number of serial communication standards and protocols.

[4] Yong Guo, Yubo Wang, Xiaoke Tang, 2020, "A SPI Interface Module Verification Method Based on UVM", IEEE.

The serial peripheral interface (SPI) is an important module for realizing communication between the APB bus in the SOC chip and peripheral SPI devices. Therefore, efficient, and sufficient verification of the function of the SPI module is very important for the design and manufacture of the SOC chip. In this article, a verification environment for SPI module is built based on universal verification methodology (UVM). By introducing the register abstraction layer (RAL), the efficiency of register attribute checking and configuration is greatly improved. In addition, the use of constrained random excitation and automatic result comparison function has realized the full verification of the SPI function, with a coverage rate of 100%. Finally, the verification environment of the SPI module has been successfully migrated to the SOC verification environment.

[5] Dvijen Trivedi; Aniruddha Khade; Kashish Jain; Ruchira Jadhav,2018, "SPI to I2C Protocol Conversion Using Verilog",IEEE.

The purpose of this paper is to design and simulate a Protocol Conversion Unit (PCU) for seamless communication between the two widely accepted serial communication protocols SPI and I2C. Design given in this paper takes data from a sender device working on SPI protocol and sends it to a receiver device working on I2C protocol, which otherwise without such design would not be possible. SPI supports full duplex communication unlike I2C which is half duplex. Also, SPI is faster than I2C. I2C on the other hand is just a two-wire interface as unlike SPI it does not use a dedicated Slave Select line. Instead I2C relies on Address and Acknowledgement scheme to communicate with slave. Thus, in areas where the controlling device needs to communicate with a lot of peripheral devices, it is essential for the controller to send commands and data to the concerned peripheral device quickly using high

speed of SPI and at the same time save on dedicated pins for each peripheral device using a rather simple Two Wire Interface of I2C, a design capable of providing conversion between SPI and I2C formats becomes essential. In this paper support for just one peripheral device is given. The design in this paper can be upgraded to support large no of peripherals by providing a First in First Out (FIFO) Queue for storing commands and data along with corresponding addresses of peripheral devices in the Protocol Conversion Unit (PCU).

[6] M. Poorani; R. Kurunjimalar,2016, "Design implementation of UART and SPI in single FGPA", IEEE.

The UART (Universal Asynchronous Receiver and transmitter) controller is the key component of the serial communications subsystem of a computer. The UART takes bytes of data and transmits the individual bits in a sequential fashion. SPI is a common technology used nowadays for communication with peripheral devices where we want to transfer data speedily and within real time constraints. In the existing work data acquisition system for underground has been designed for counters triggered by surface detectors and they have used UART and SPI protocol for the communication with dedicated processor. In this paper we have used Xilinx 12.4i and SPARTAN 3E FPGA to implement the whole system. The hyper terminal is used to check the UART protocol and 12-bit DAC MCP4922 used for check SPI protocol. The proposed work can provide both protocols effectively for the wireless serial communication. These serial protocols mainly used in Zigbee wireless technology.

# SERIAL PERIPHERAL INTERFACE – AN OVERVIEW

# CHAPTER 3

# SERIAL PERIPHERAL INTERFACE – AN OVERVIEW

## 3.1 INTERFACE

The SPI bus specifies four logic signals:

- SCLK: Serial Clock (output from master)
- MOSI: Master Out Slave In (data output from master)
- MISO: Master In Slave Out (data output from slave)
- CS /SS: Chip/Slave Select (often active low, output from master to indicate that data is being sent)

MOSI on a master connects to MOSI on a slave. MISO on a master connects to MISO on a slave. Slave Select has the same functionality as chip select and is used instead of an addressing concept.

The signal names above can be used to label both the master and slave device pins as well as the signal lines between them in an unambiguous way, and are the most common in modern products. Pin names are always capitalized e.g., "Chip Select," not "chip select."

**Serial Clock:**

- SCK, SCLK, CLK, SCL

**Master Output → Slave Input (MOSI):**

- SIMO, MTSR - correspond to MOSI on both master and slave devices, connects to each other
- SDI, DI, DIN, SI - on slave devices; connects to MOSI on master, or to below connections
- SDO, DO, DOUT, SO - on master devices; connects to MOSI on slave, or to above connections.

**Master Input ← Slave Output (MISO):**

- SOMI, MRST - correspond to MISO on both master and slave devices, connects to each other
- SDO, DO, DOUT, SO - on slave devices; connects to MISO on master, or to below connections
- SDI, DI, DIN, SI - on master devices; connects to MISO on slave, or to above connections

**Slave Select:**

- SS, SS, SSEL, NSS, /SS, SS# (slave select)
- CS, CS (chip select)
- CSN (chip select/enable)
- CE (chip enable)

### 3.2 OPERATION

The SPI bus can operate with a single master device and with one or more slave devices.

If a single slave device is used, the SS pin *may* be fixed to logic low if the slave permits it. Some slaves require a falling edge of the chip select signal to initiate an action. An example is the Maxim MAX1242 ADC, which starts conversion on a high→low transition. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (*electrically disconnected*) when the device is not selected. Devices without tri-state outputs cannot share SPI bus segments with other devices without using an external tri-state buffer.

### 3.2.1 DATA TRANSMISSION

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz. The master then selects the slave device with a logic level 0 on the select line. If a waiting period is required, such as for an analog-to-digital conversion, the master must wait for at least that period before issuing clock cycles.

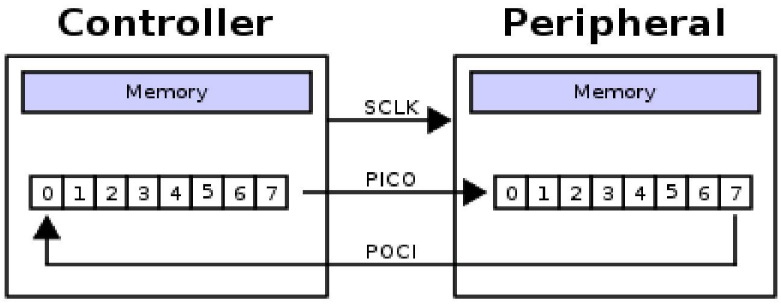



**Figure 3.1** Hardware setup using two shift registers

During each SPI clock cycle, a full-duplex data transmission occurs. The master sends a bit on the

MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

Transmissions normally involve two shift registers of some given word-size, such as eight bits, one in the master and one in the slave; they are connected in a virtual ring topology. Data is usually shifted out with the most significant bit first. On the clock edge, both master and slave shift out a bit and output it on the transmission line to the counterpart. On the next clock edge, at each receiver the bit is sampled from the transmission line and set as a new least-significant bit of the shift register. After the register bits have been shifted out and in, the master and slave have exchanged register values. If more data needs to be exchanged, the shift registers are reloaded and the process repeats. Transmission may continue for any number of clock cycles. When complete, the master stops toggling the clock signal, and typically deselects the slave.

### 3.2.2 CLOCK POLARITY AND PHASE

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Motorola SPI Block Guide names these two options as CPOL and CPHA (for clock polarity and phase) respectively, a convention most vendors have also adopted.

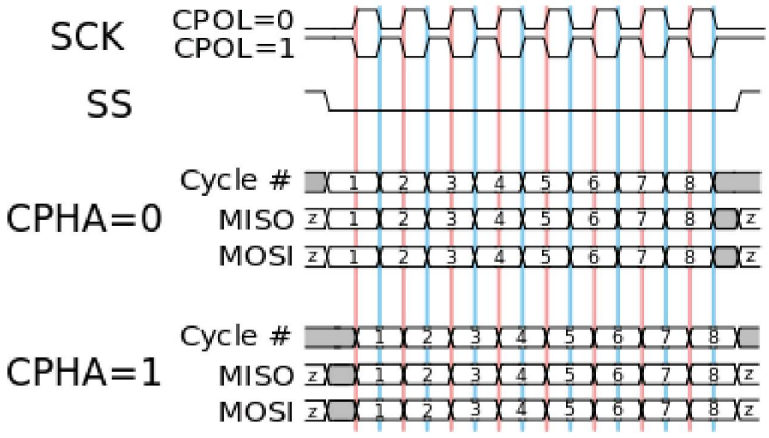



**Figure 3.2** Timing diagram of CLOCK PHASE and CLOCK POLARITY

The timing is further described below and applies to both the master and the slave device.

- CPOL determines the polarity of the clock. The polarities can be converted with a simple inverter.
  - CPOL=0 is a clock which idles at 0, and each cycle consists of a pulse
  - of 1. That is, the leading edge is a rising edge, and the trailing edge is a falling edge.
  - CPOL=1 is a clock which idles at 1, and each cycle consists of a pulse of 0. That is, the leading edge is a falling edge, and the trailing edge is a rising edge.
- CPHA determines the timing (i.e., phase) of the data bits relative to the clock pulses. Conversion between these two forms is non-trivial.

  - For CPHA=0, the "out" side changes the data on the trailing edge of the preceding clock cycle, while the "in" side captures the data on (or shortly after) the leading edge of the clock cycle. The outside holds the

data valid until the trailing edge of the current clock cycle. For the first cycle, the first bit must be on the MOSI line before the leading clock edge. An alternative way of considering it is to say that a CPHA=0 cycle consists of a half cycle with the clock idle, followed by a half cycle with the clock asserted.

- For CPHA=1, the "out" side changes the data on the leading edge of the current clock cycle, while the "in" side captures the data on (or shortly after) the trailing edge of the clock cycle. The outside holds the data valid until the leading edge of the following clock cycle. For the last cycle, the slave holds the MISO line valid until slave select is deserted. An alternative way of considering it is to say that a CPHA=1 cycle consists of a half cycle with the clock asserted, followed by a half cycle with the clock idle.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle.

This adds more flexibility to the communication channel between the master and slave.

### 3.2.3 MODE NUMBERS

The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

| Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

**Table 3.1** SPI Modes

Another commonly used notation represents the mode as a (CPOL, CPHA) tuple; e.g., the value '(0, 1)' would indicate CPOL=0 and CPHA=1.

### 3.2.4 INDEPENDENT SLAVE CONFIGURATION

In the independent slave configuration, there is an independent chip select line for each slave. This is the way SPI is normally used. The master asserts only one chip select at a time.

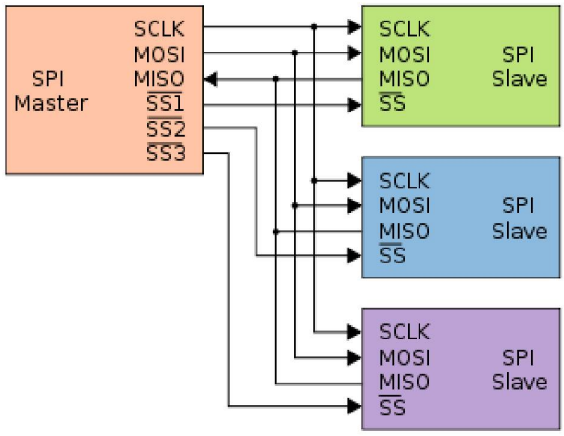



**Figure 3.3** Typical SPI bus: master and three independent slaves

Pull-up resistors between power source and chip select lines are recommended for systems where the master's chip select pins may default to an undefined state. When separate software routines initialize each chip select and communicate with its slave, pull-up resistors prevent other uninitialized slaves from responding.

Since the MISO pins of the slaves are connected, they are required to be tri-state pins (high, low or high-impedance), where the high-impedance output must be applied when the slave is not selected. Slave devices not supporting tri-state may be used in independent slave configuration by adding a tri-state buffer chip controlled by the chip select signal. (Since only a single signal line needs to be tristated per slave, one typical standard logic chip that contains four tristate buffers with independent gate inputs can be used to interface up to four slave devices to an SPI bus)

### 3.2.5 DAISY CHAIN CONFIGURATION

Some products that implement SPI may be connected in a daisy chain configuration, the first slave output being connected to the second slave input, etc.

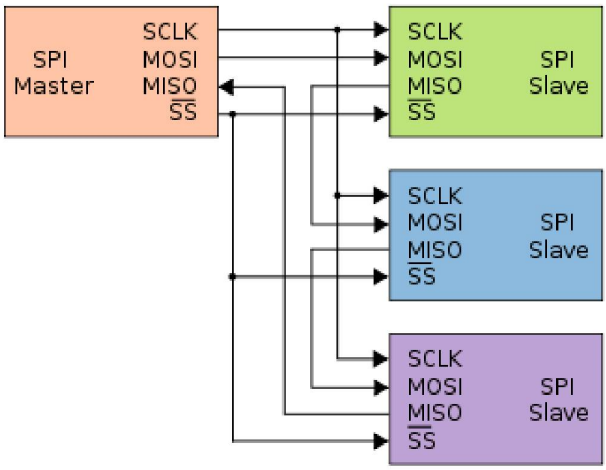



**Figure 3.4** Daisy-chained SPI bus: master and cooperative slaves

The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of the data it received during the first group of clock pulses. The whole chain acts as a communication shift register; daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Each slave copies input to output in the next clock cycle until active low SS line goes high. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.

### 3.2.6 ADVANTAGES

- Full duplex communication in the default version of this protocol
- Push-pull drivers (as opposed to open drain) provide good signal integrity and high speed
- Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speed
- Complete protocol flexibility for the bits transferred
  - Not limited to 8-bit words

- Arbitrary choice of message size, content, and purpose

- Extremely simple hardware interfacing
  - Typically, lower power requirements than I²C or SMBus due to less circuitry (including pull up resistors)
  - No arbitration or associated failure modes - unlike CAN-bus
  - Slaves use the master's clock and do not need precision oscillators
  - Slaves do not need a unique address – unlike I²C or GPIB or SCSI
  - Transceivers are not needed - unlike CAN-bus
- Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
- At most one unique bus signal per device (chip select); all others are shared
- Signals are unidirectional allowing for easy galvanic isolation
- Simple software implementation

### 3.2.7 DISADVANTAGES

- Requires more pins on IC packages than I²C, even in the *three-wire* variant
- No in-band addressing; out-of-band chip select signals are required on shared buses
- Extensibility severely reduced when multiple slaves using different SPI Modes are required. Access is slowed down when master frequently needs to reinitialize in different modes.
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
- Typically supports only one master device (depends on device's hardware implementation)
- No error-checking protocol is defined
- Without a formal standard, validating conformance is not possible
- Only handles short distances compared to RS-232, RS-485, or CAN-bus. (Its distance can be extended with the use of transceivers like RS-422.)

- Opto-isolators in the signal path limit the clock speed for MISO transfer because of the added delays between clock and data
- Many existing variations, making it difficult to find development tools like host adapters that support those variations
- SPI does not support hot swapping (dynamically adding nodes).
- Interrupts must either be implemented with out-of-band signals or be faked by using periodic polling similarly to USB 1.1 and 2.0.
- Some variants like dual SPI, quad SPI, and three-wire serial buses defined below are half-duplex.

### 3.2.8 APPLICATIONS

The board real estate savings compared to a parallel I/O bus are significant, and have earned SPI a solid role in embedded systems. That is true for most system-on-a-chip processors, both with higher end 32-bit processors such as those using ARM, MIPS, or PowerPC and with other microcontrollers such as the AVR, PIC, and MSP430. These chips usually include SPI controllers capable of running in either master or slave mode. In-system programmable AVR controllers (including blank ones) can be programmed using a SPI interface.

Chip or FPGA based designs sometimes use SPI to communicate between internal components; on-chip real estate can be as costly as its on-board cousin.

The full-duplex capability makes SPI very simple and efficient for single master/single slave applications. Some devices use the full-duplex mode to implement an efficient, swift data stream for applications such as digital audio, digital signal processing, or telecommunications channels, but most off-the-shelf chips stick to half-duplex request/response protocols.

SPI is used to talk to a variety of peripherals, such as

- Sensors: temperature, pressure, ADC, touchscreens, video game controllers
- Control devices: audio codecs, digital potentiometers, DAC

- Camera lenses: Canon EF lens mount
- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
- Memory: flash and EEPROM
- Real-time clocks
- LCD, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant)

For high-performance systems, FPGAs sometimes use SPI to interface as a slave to a host, as a master to sensors, or for flash memory used to bootstrap if they are SRAM-based.

Although there are some similarities between the SPI bus and the JTAG (IEEE 1149.1-2013) protocol, they are not interchangeable. The SPI bus is intended for high-speed, on-board initialization of device peripherals, while the JTAG protocol is intended to provide reliable test access to the I/O pins from an off board controller with less precise signal delay and skew parameters. While not strictly a level sensitive interface, the JTAG protocol supports the recovery of both setup and hold violations between JTAG devices by reducing the clock rate or changing the clock's duty cycles. Consequently, the JTAG interface is not intended to support extremely high data rates.

# SOFTWARE TOOLS USED

# CHAPTER 4

# SOFTWARE TOOLS USED

## 4.1 QUARTUS II 10.0 WEB EDITION

The Altera Quartus II design software provides a complete, multiplatform design environment that easily adapts to your specific design needs. It is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes solutions for all phases of FPGA and CPLD design. (Figure 4.2)


**Figure 4.1** Quartus II Graphical User Interface

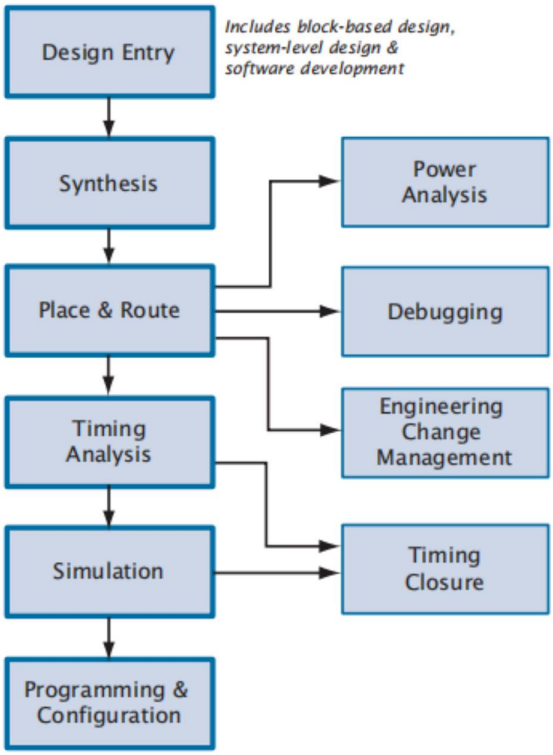



**Figure 4.2** RTL Design Flow

The Quartus II software allows you to use the Quartus II graphical user interface and command-line interface for each phase of the design flow. You can use one of these interfaces for the entire flow, or you can use different options at different phases.

The Quartus II software includes a modular Compiler. The Compiler includes the following modules (modules marked with an asterisk are optional during a compilation, depending on your settings)

- Analysis & Synthesis
- Partition Merge
- Fitter
- Assembler
- TimeQuest Timing Analyzer
- Design Assistant
- EDA Netlist Writer
- HardCopy Netlist Writer

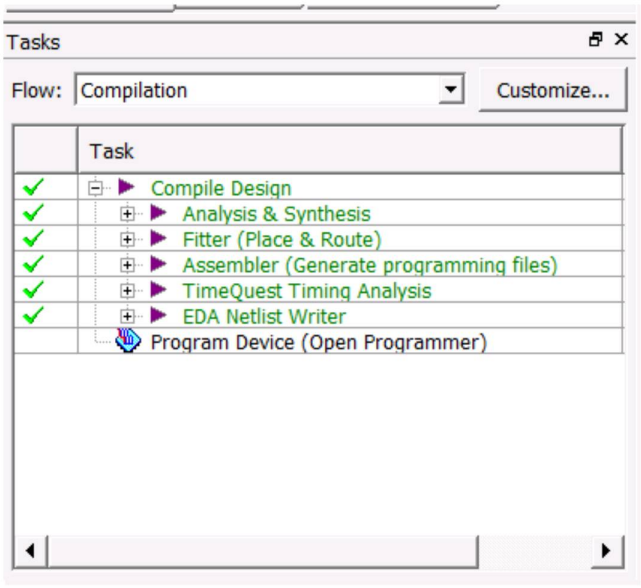



**Figure 4.3** Task Window

To run all Compiler modules as part of a full compilation, on the Processing menu, click Start Compilation. You can also run each module individually by pointing to Start on the Processing menu, and then clicking the command for the module you want to start. In addition, you can use the Tasks window to start Compiler modules individually. The Tasks window also allows you to change settings or view the report file for the module, or to start other tools related to each stage in a flow.

## 4.2 MODELSIM ALTERA

ModelSim is a multi-language environment by Siemens (previously developed by Mentor Graphics,) for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Intel Quartus Prime, PSIM, Xilinx ISE or Xilinx Vivado.Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

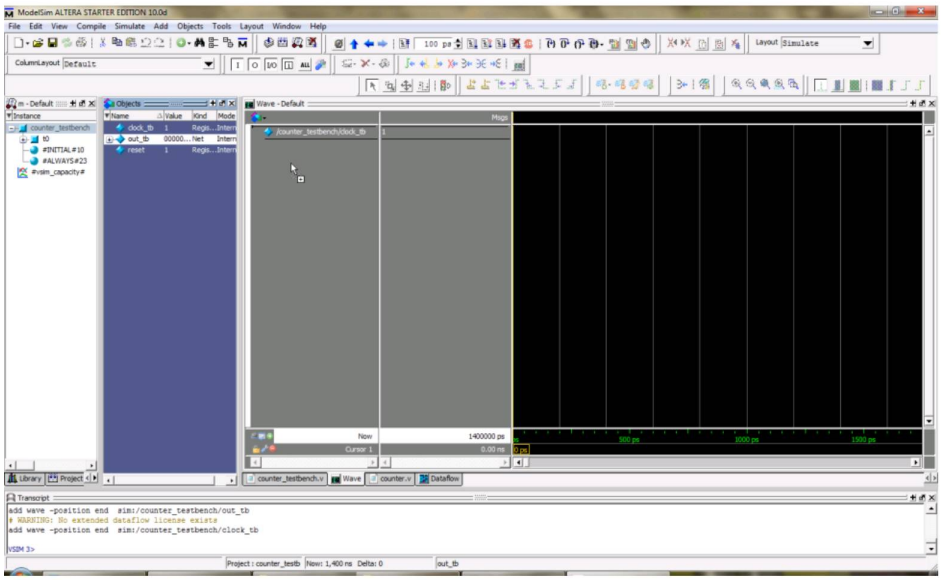


**Figure 4.4** Modelsim Altera User Interface

Altera- specific modeling software includes Altera-specific timing simulations as well as VHDL or Verilog HDL simulation and testbenches for Altera PLDs. This document describes ModelSim Altera version 5.6a, as well as ModelSim PE version 5.6.

**Language support:**

ModelSim uses a unified kernel for simulation of all supported languages, and the method of debugging embedded C code is the same as VHDL or Verilog.

ModelSim and Questa Sim products enable simulation, verification and debugging for the following languages

- VHDL
- Verilog
- Verilog 2001
- SystemVerilog
- PSL
- SystemC

# DESIGN METHODOLOGY

# CHAPTER 5

# DESIGN METHODOLOGY

## 5.1 BLOCK DIAGRAM

Figure 5.1 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.

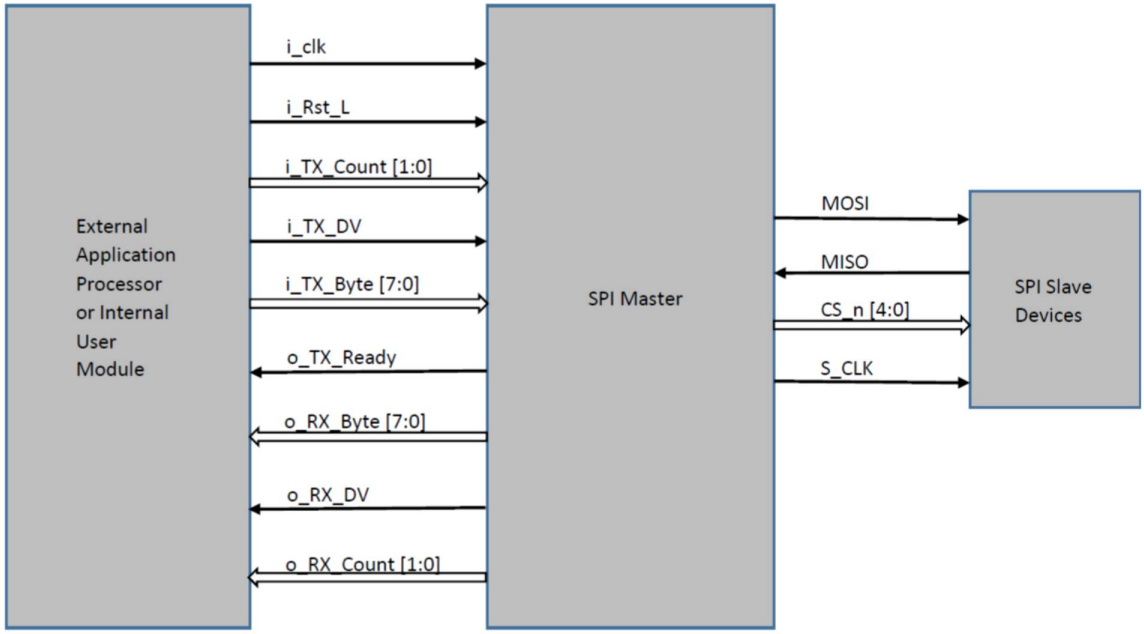



**Figure 5.1** Block Diagram of SPI Model

## 5.2 SIGNAL DESCRIPTIONS

| Signal | Width | Type | Description |
|---|---|---|---|
| i_clk | 1 | Input | System Clock. |
| i_Rst_L | 1 | Input | Asynchronous active low reset. |
| i_TX_DV | 1 | Input | Asserted momentarily to begin SPI transaction.<br>Active High. |
| i_TX_Byte [7:0] | 8 | Input | Parallel input data from the processor interface.<br>Captured data is sent to MOSI line. |
| o_TX_Ready | 1 | Output | Positive strobe to indicate that data is captured from the i_TX_Byte port. |
| o_RX_Byte [7:0] | 8 | Output | Parallel output data sent to the processor interface.<br>Data from this port comes from the MOSI line. |
| o_RX_DV | 1 | Output | Positive strobe to indicate that data can be read from the o_RX_Byte port. |
| i_TX_Count [7:0] | 8 (customizable) | Input | Sets the number of bytes in the SPI transaction (Master to slave) |
| o_RX_Count [7:0] | 8 (customizable) | Output | Sets the number of bytes in the SPI transaction (Slave to master) |
| MOSI | 1 | Output | SPI data bus – master in slave out |
| MISO | 1 | Input | SPI data bus – master out slave in |
| CS_N [4:0] | 5 | Output | SPI slave select outputs<br>Active low |
| SCLK | 1 | Output | SPI serial clock |

**Table 5.1** Signal Descriptions

## 5.3 SPI_MASTER_CONTROLLER

The SPI_Master_Controller is the top-level module. This module has two inner modules SPI_Master_With_Single_CS and SPI_Slave. It connects the SPI_Master and SPI_Slave through four wires. It also gives inputs to the Master as well as Slave for the data transfer.

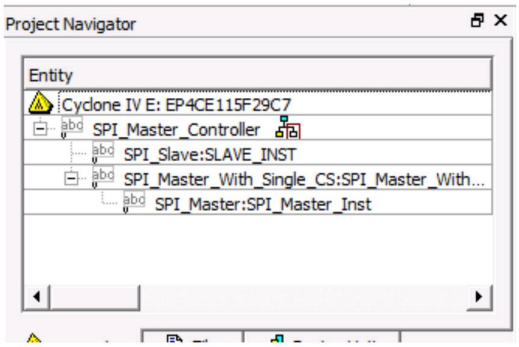



**Figure 5.2** Hierarchy of Verilog Modules.

It has the following input lines from the system:

- i_Clk
- i_Rst_L
- i_TX_Count
- i_DV_To_Master
- i_DV_To_Slave
- i_input_To_Master
- i_input_To_Slave

It has the following lines as output from the system:

- o_RX_Count
- o_output_From_Master
- o_output_From_Slave
- o_DV_From_Master
- o_DV_From_Slave

It has the following wires which connects the Master and the Slave:

- MOSI
- MISO
- SCLK
- CS

### 5.4 SPI_MASTER

This is the lowest module for the Master which consists of all the signals that needed for data transmission. It sends data given by the system to the slave. It also accepts the data from the selected slave and give it to the system.

It has the following input signals:

- i_Clk
- i_Rst_L
- i_TX_Byte
- i_TX_DV
- i_SPI_MISO

It has the following output signals:

- o_TX_Ready
- o_RX_DV
- o_RX_Byte
- o_SPI_Clk
- o_SPI_MOSI

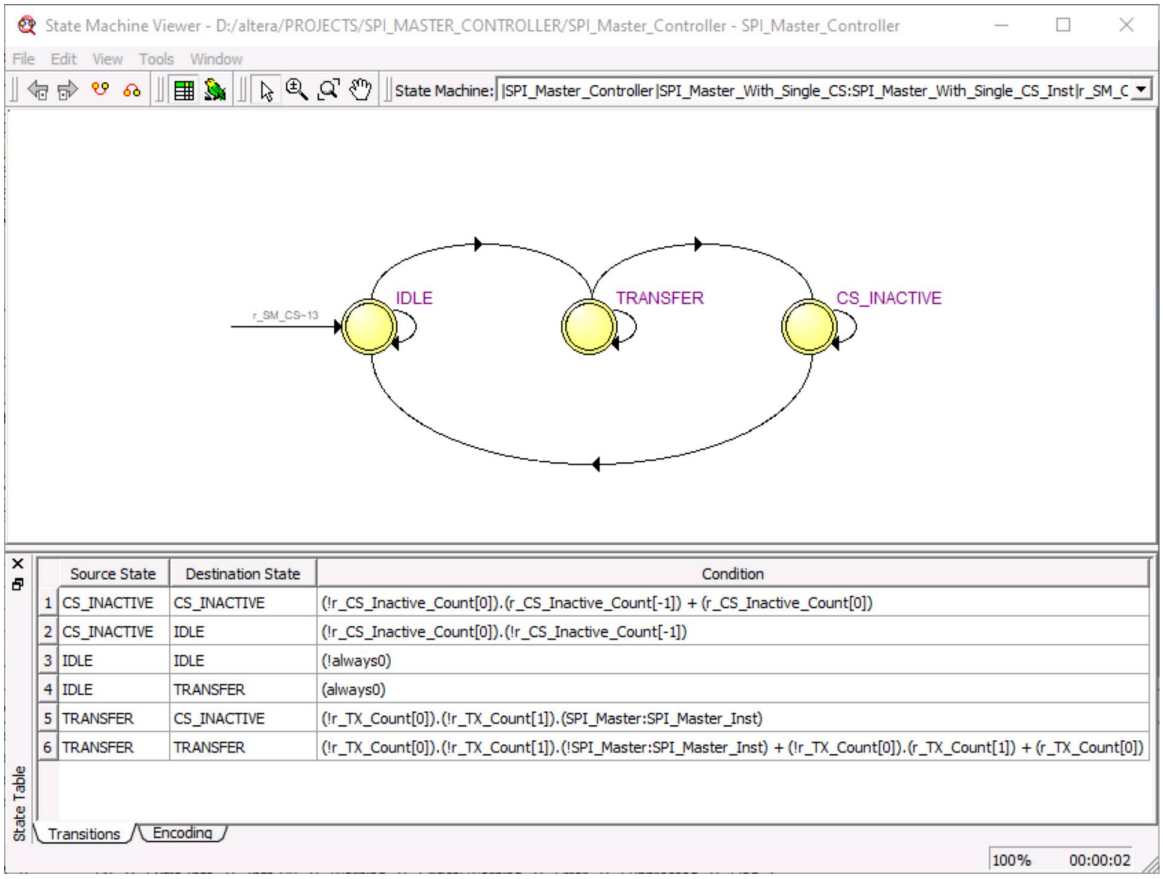
The SPI_Master works as follows,

- Whenever there is a data valid, the master wakes up.
- It stores whatever the data is present in the i_TX_Byte bus when the i_TX_DV is asserted.
- After, when the i_TX_DV goes low, the master starts transmitting the data stored bit by bit to the Slave through o_SPI_MOSI in terms of S_Clock cycles.
- After completing the transmission, the master sends a HIGH to the system through o_TX_Ready that it has successfully transmitted the given byte and it is ready for the next byte for transmission.
- The Master also received data from Slave bit by bit through o_SPI_MISO in terms of S_Clock cycles.
- The Master collects these bits in an internal register and once when a byte is collected it given to the system through o_RX_Byte along with a pulse o_RX_DV.

### 5.5 SPI_MASTER_WITH_SINGLE_CS:

This is the Main module for the Master. It contains the lowest module SPI_Master. The lowest module “SPI_Master” is responsible for transmission while this parent module is responsible for selecting the Slave. Hence, this module together with its child module, roles the play of Master.

The SPI_Master_With_Single_CS works as a STATE MACHINE




| | Source State | Destination State | Condition |
|---|---|---|---|
| 1 | CS_INACTIVE | CS_INACTIVE | (!r_CS_Inactive_Count[0]).(r_CS_Inactive_Count[-1]) + (r_CS_Inactive_Count[0]) |
| 2 | CS_INACTIVE | IDLE | (!r_CS_Inactive_Count[0]).(!r_CS_Inactive_Count[-1]) |
| 3 | IDLE | IDLE | (!always0) |
| 4 | IDLE | TRANSFER | (always0) |
| 5 | TRANSFER | CS_INACTIVE | (!r_TX_Count[0]).(!r_TX_Count[1]).(SPI_Master:SPI_Master_Inst) |
| 6 | TRANSFER | TRANSFER | (!r_TX_Count[0]).(!r_TX_Count[1]).(!SPI_Master:SPI_Master_Inst) + (!r_TX_Count[0]).(r_TX_Count[1]) + (r_TX_Count[0]) |

**Figure 5.3** State Diagram of SPI_Master_With_Single_CS

- The Master will initially be IDLE state.
- When a data valid pulse is received through i_TX_DV, then the Master goes from IDLE state to TRANSFER state.
- In this State, the Master selects a Slave and starts transferring data to that slave and accepting data from the selected slave.
- After successful transaction, the Master goes to the CS_INACTIVE state.
- In this State, the Master is never transmitting data nor accepting data. After some i_Clk cycles, the Master will again go to the IDLE State.

## 5.6 SPI_SLAVE

The SPI_Slave is the Module contained in the SPI_Master_Controller and performs the function of Slave in the SPI Communication Protocol. Each Slave may be contained in different peripherals for the communication with the main system. Not all slaves can communicate with all masters. Only compatible slaves alone communicate with their respective Masters.

The SPI_Slave works as follows:

- The Slave remains in halt state until an Active LOW Slave Select or Chip Select signal is asserted through CS.
- Once the Slave turns ON, it will receive data from Master through MOSI, bit by bit and once a byte is collected in an internal register, the slave sends the byte to the peripheral which the slave is belonged to.
- The Slave also gets data from its peripheral and sends it to the Master bit by bit through MISO.

## 5.7 PROCESSOR INTERFACE TIMING DIAGRAM

The following describes the timing for the Processor Interface as illustrated by Figure 5.2. 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.

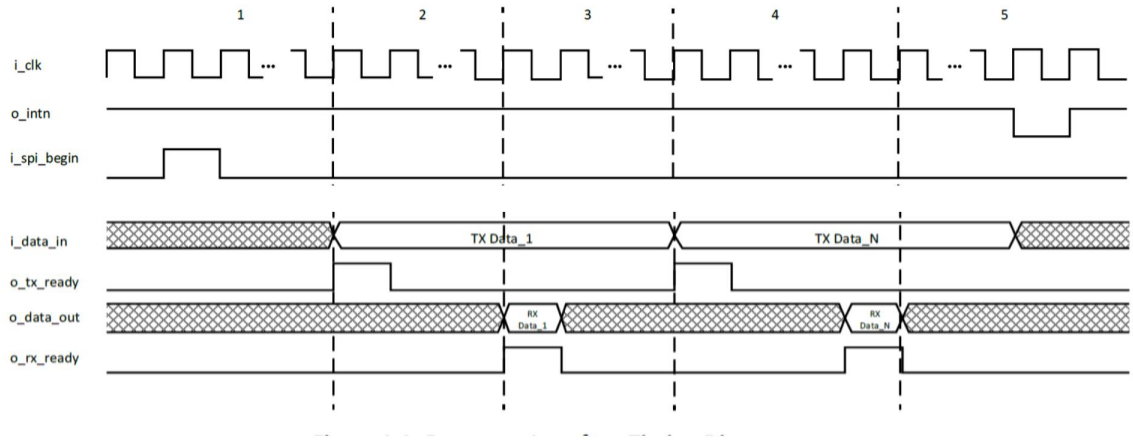



**Figure 5.4** Processor Interface Timing Diagram

1. The Reference Design starts in an idle state. The Internal User Module or External Application Processor can prepare the inputs for the i_Clk, i_Rst, i_TX_Count ports as described in Table 5.1. Afterwards, the i_TX_DV 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_RX_Byte port should be ready. During this point, each bit of data is captured and subsequently sent to the MOSI SPI port. The i_RX_Byte 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_RX_Byte 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, steps 2 and 3 are automatically repeated until the defined number of bytes is reached.
5. When the total number of bytes defined is reached, an interrupt is generated and the reference design returns to an idle state.

## 5.8 SPI INTERFACE TIMING DIAGRAM

Figure 5.3 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).

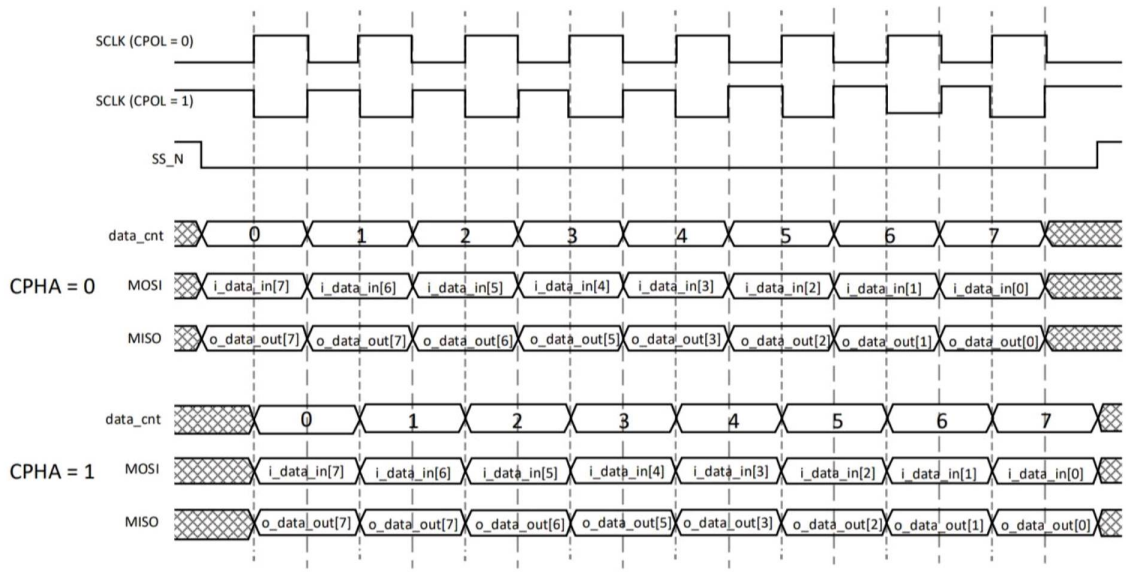



**Figure 5.5** SPI Interfacing Diagram (MSB First)

This reference design pushes bits of data on the MOSI line from a shift register based on bit count (i_TX_Count) 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.

## 5.9 CUSTOMIZATION

| Category | Compiler Directives | Description |
|---|---|---|
| SPI Mode | SPI_MODE | To enable a defined CPHA and CPOL |
| Clock Cycle Selection | CLKS_PER_HALF_BIT | Defines how many i_clk cycles invested for a single bit transfer in half. |
| Data Width | MAX_BYTES_PER_CS | Defines the width of data transfer in SPI transaction in bytes |
| SPI Transaction Width | CS_INACTIVE_CLKS | Defines the interval between two SPI transaction in terms of i_clk cycles |

**Table 5.2** Compiler Directive Options

## 5.10 PACKAGED DESIGN

The design folder (SPI_Master_Controller) contains four subfolders: .sopcBuilder, db, incremental db, simulation. The details of each subfolder are as follows:

**SOPC Builder** – contains the links to literature documents, manuals, tutorials, user guides and training classes.

**Simulation** – contains the modelsim altera folder - contains the simulation file (.vsim) used to run RTL simulation on ModeSim Altera.

**Incremental db and db** – contains the project database files and they can be deleted.

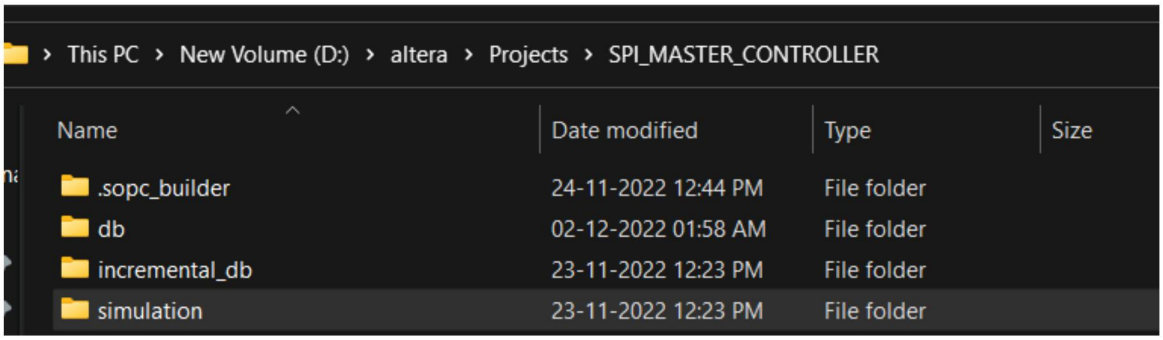



**Figure 5.6** Packaged Design Directory Structure

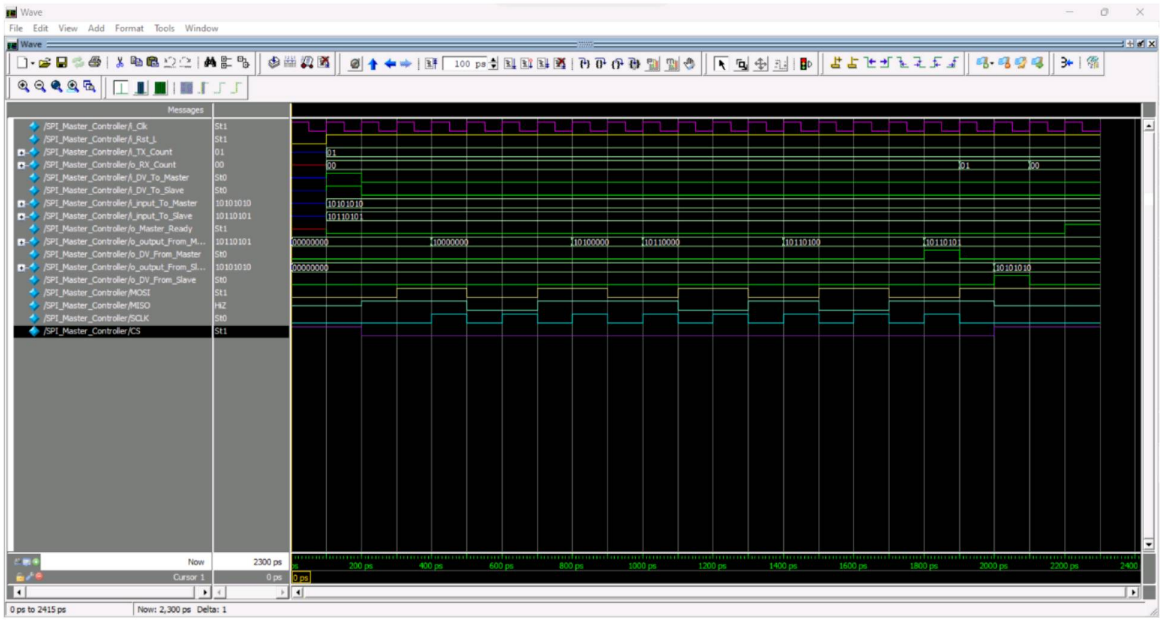
# OUTPUTS AND RESULTS

# CHAPTER 6

# OUTPUTS AND RESULTS

## 6.1 SPI MODE 0

Initially, the reset is given along the "i_Rst_L" line. The input to the Master and the Slave is given in terms of bytes. The length of this data can be changed from bytes to any other arbitrary length by changing the parameters. The Data Valid signals are also given together with the input data. Once the data valid signals are received, the transmission starts by generating serial clock and enabling Slave Select or Chip Select. In this mode 0, the bits are sampled at the RISING Edge and a new bit is supplied at the FALLING Edge. After completing the transaction, the master goes to inactive state for some clock cycles and an active 'HIGH' signal is enabled to indicate that the master has successfully completed the transaction and it is ready for the next transaction. These customizations are not present in the Slave as Master has all the controls in the SPI transaction. Even the data that need to be supplied by the Slave is controlled by the Master. The control signals can be sent to the Slave through the MOSI bus prior to the transaction. The control signal can be of any byte or arbitrary length word that can be set by user who needs to use the Master-Slave configuration.



## 6.2 SPI MODE 1

Initially, the reset is given along the "i_Rst_L" line. The input to the Master and the Slave is given in terms of bytes. The length of this data can be changed from bytes to any other arbitrary length by changing the parameters. The Data Valid signals are also given together with the input data.

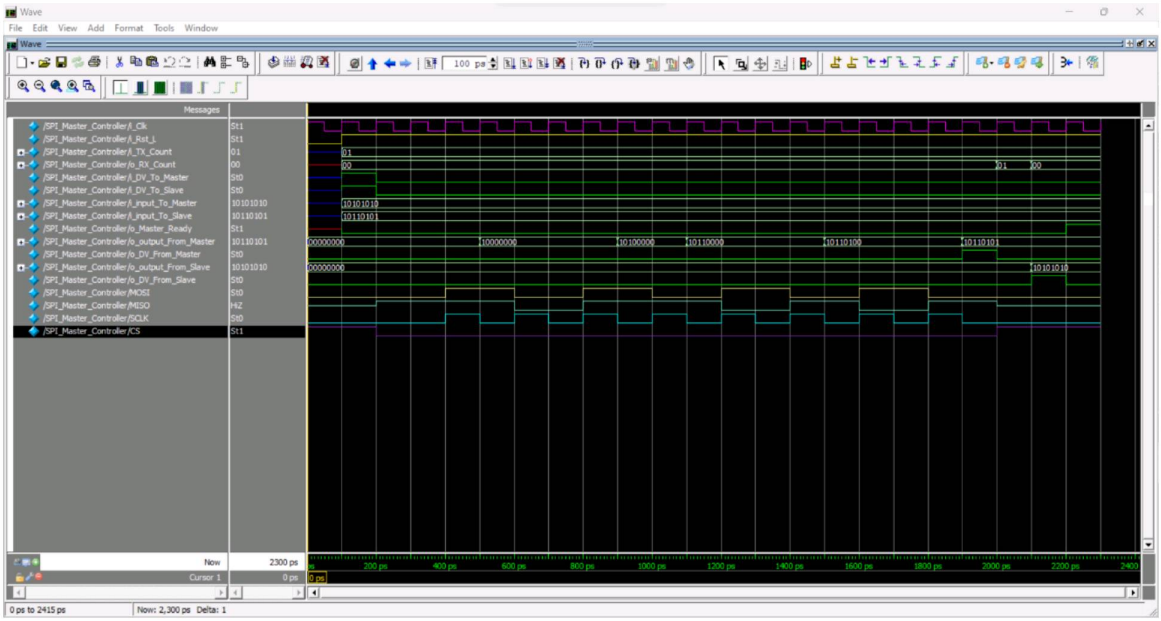


**Figure 6.2** SPI MODE 1 OUTPUT

Once the data valid signals are received, the transmission starts by generating serial clock and enabling Slave Select or Chip Select. The bits are sampled at the FALLING Edge and a new bit is supplied at the RISING Edge. After completing the transaction, the master goes to inactive state for some clock cycles and an active 'HIGH' signal is enabled to indicate that the master has successfully completed the transaction and it is ready for the next transaction.

### 6.3 SPI MODE 2

Initially, the reset is given along the "i_Rst_L" line. The input to the Master and the Slave is given in terms of bytes. The length of this data can be changed from bytes to any other arbitrary length by changing the parameters. The Data Valid signals are also given together with the input data.

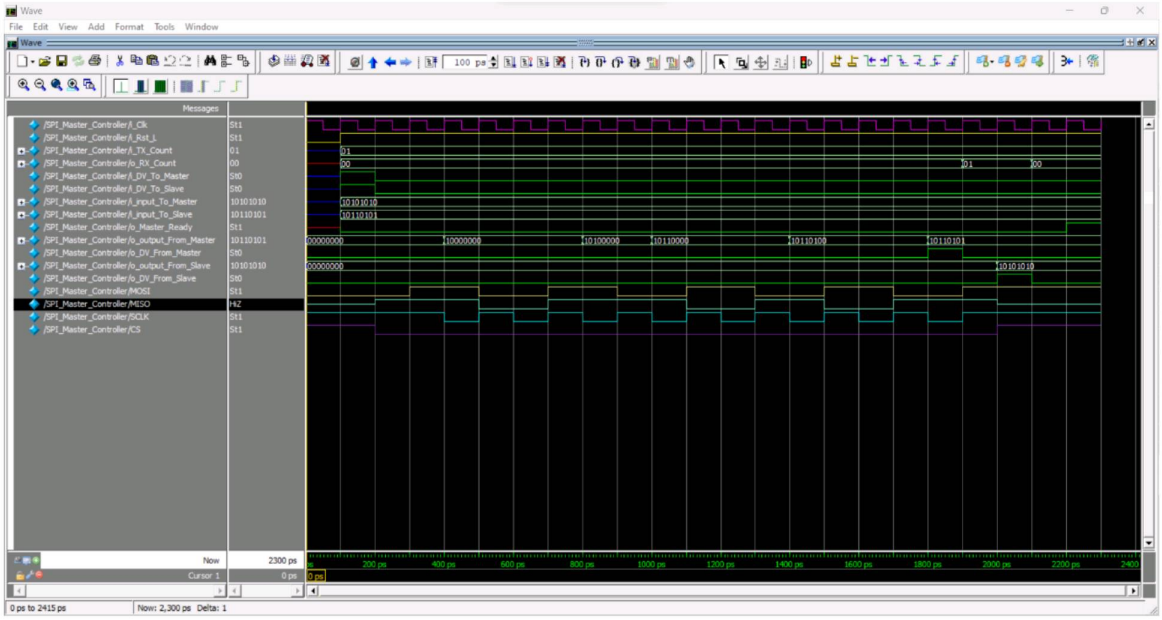


**Figure 6.3** SPI MODE 2 OUTPUT

Once the data valid signals are received, the transmission starts by generating serial clock and enabling Slave Select or Chip Select. The bits are sampled at the FALLING Edge and a new bit is supplied at the RISING Edge. After completing the transaction, the master goes to inactive state for some clock cycles and an active 'HIGH' signal is enabled to indicate that the master has successfully completed the transaction and it is ready for the next transaction.

## 6.4 SPI MODE 3

Initially, the reset is given along the "i_Rst_L" line. The input to the Master and the Slave is given in terms of bytes. The length of this data can be changed from bytes to any other arbitrary length by changing the parameters. The Data Valid signals are also given together with the input data.

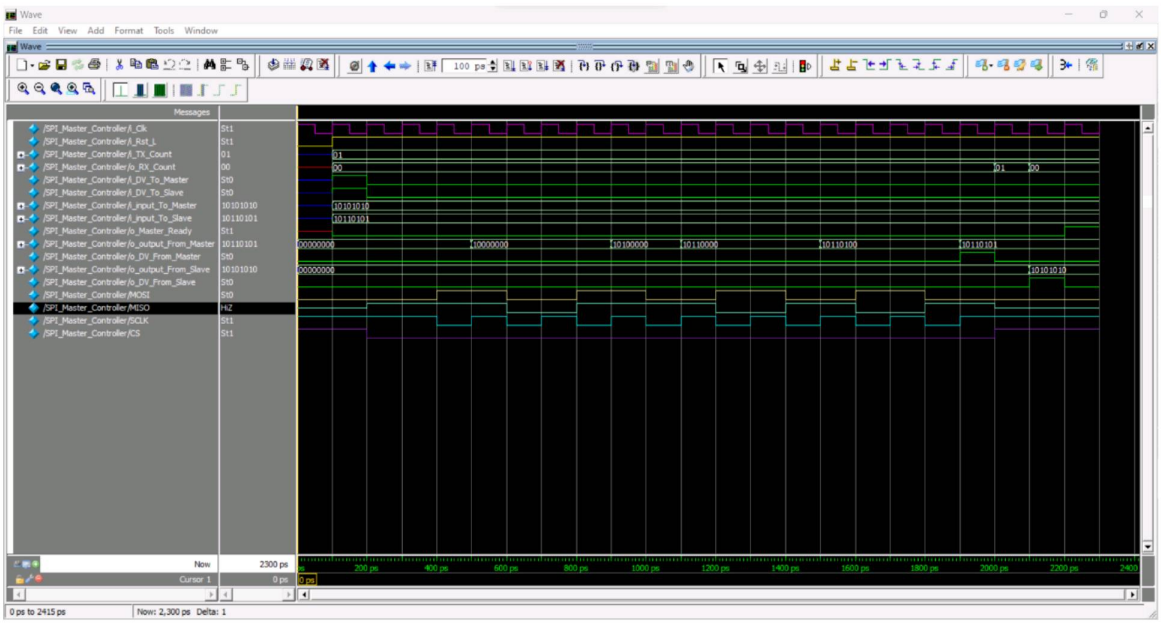


**Figure 6.4** SPI MODE 3 OUTPUT

Once the data valid signals are received, the transmission starts by generating serial clock and enabling Slave Select or Chip Select. The bits are sampled at the RISING Edge and a new bit is supplied at the FALLING Edge. After completing the transaction, the master goes to inactive state for some clock cycles and an active 'HIGH' signal is enabled to indicate that the master has successfully completed the transaction and it is ready for the next transaction.

# REFERENCES

# CHAPTER 7

# REFERENCES


[1] Karamalaputti, P., Kumar, A, 2022, “Design and Simulation of Serial Peripheral Interface Protocol Using Pulsed Latches”. First International Conference on Computational Electronics for Wireless Communications. Lecture Notes in Networks and Systems, Springer, Singapore, vol 329.

[2] Orhan Gazi & A. Çağrı Arlı, 2021, “Serial Peripheral Interface. In: State Machines using VHDL” Springer, Cham.

[3] Dawoud Shenouda Dawoud, Peter Dawoud, 2020, “Serial Peripheral Interface (SPI),” IEEE, River Publishers, 191 – 244.

[4] Yong Guo, Yubo Wang, Xiaoke Tang, 2020, “A SPI Interface Module Verification Method Based on UVM”, IEEE.

[5] Dvijen Trivedi; Aniruddha Khade; Kashish Jain; Ruchira Jadhav,2018, “SPI to I2C Protocol Conversion Using Verilog”,IEEE.

[6] M. Poorani; R. Kurunjimalar,2016, “Design implementation of UART and SPI in single FGPA”, IEEE.

[7] F. Leens, February 2009 “An Introduction to I2C and SPI Protocols,” IEEE Instrumentation & Measurement Magazine, pp. 8-13.

[8] F. Leens, June 2008. “Solutions for SPI Protocol Testing and Debugging in Embedded System,” Byte Paradigm’s White Paper, pp. 1-9, Revision 1.00.

[9] Motorola Inc., “SPI Block Guide V03.06,” February 2003.

[10] A.K. Oudjida et al, “FPGA Implementation of I2C and SPI Protocols: A Comparative Study”. Proceedings of the 16th edition of the IEEE International Conference on Electronics Circuits and Systems ICECS, pp.507 -510, ISBN: 978-1-4244-5091-6, December 13-16 2009, Yasmine Hammamet, Tunisia.

[11] July 2006. “OPB Serial Peripheral Interface (SPI) (V1.00e),” Xilinx Logicore, DS464.

[12] Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition by Samir Palnitkar.

[13] December 2020 Generic Soft SPI Master Controller Reference Design FPGA-RD-02209-1.0.

[14] Nugent, Stephen. May 2017 “Precision SPI Switch Configuration Increases Channel Density.” *Analog Dialogue*.

[15] Usach, Miguel, September 2015,AN-1248 Application Note: *SPI Interface*. Analog Devices, In.