Design and Implementation of SPI Module in Verilog HDL using FPGA Design Flow

[1]Rahul Jandyam, [2]Sanjay Reddy Kandi, [3]Umar Farooq Mohammad
[1,2,3]B.E. Student
Department of Electronics and Communications Engineering,
University College of Engineering, Osmania University, Hyderabad, India

Abstract— Communication has always been one of the topmost concerns for human civilization. Ever since speech has originated millennia ago, it has evolved in a variety of ways from speech to symbols to cave paintings. With the advent of modern technology, the range and efficiency of communication has increased many fold. 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 paper 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 Xilinx ISE design Suite.

Keywords— SPI, FPGA Design Flow, Verilog HDL

I. INTRODUCTION

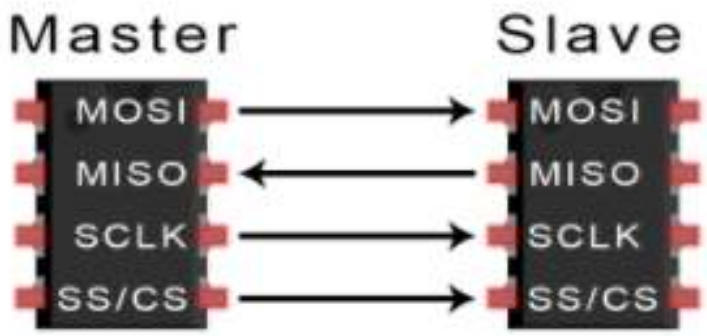
The Serial Peripheral Interface (SPI) is a short distance serial communication protocol which enables synchronous transmission of data in full duplex mode. It functions on a master – slave paradigm that is ideally suited to data stream applications. The interface was developed by Motorola in the late 1980s. Serial Peripheral Protocol (SPI) was first introduced with the first microcontroller deriving from the same architecture as the popular Motorola 68000 microprocessor, announced in 1979. Serial communication has many advantages over parallel communication like higher noise margin, high speed and less number of pins. The speed of SPI is very high (as high as 500 Mhz). The SPI can be configured either as a master or a slave. SPI module has one master and can have as many as 30 slave devices.

The Master device is usually an integrated chip (IC) or a microcontroller unit. The Slave devices can be registers, memory devices or sensors and many more. The width of the receive and transmit registers are configurable to more than sixteen bits as opposed to the conventional eight or sixteen bits.

II. SERIAL PERIPHERAL INTERFACE

SPI is also called 4 – wire protocol as it operates with 4 pins only. The SPI system is enabled by setting the SPI enable bit (SPE) in SPI control register 1. While SPE bit is set, the four associated SPI port pins are dedicated to the SPI function as :

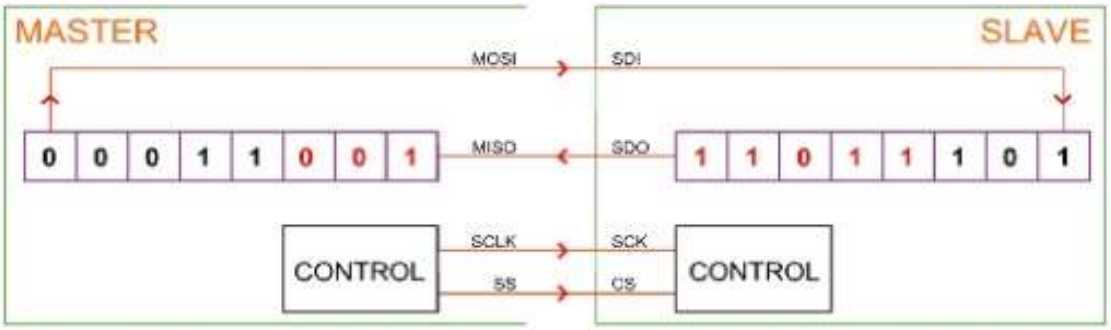
- **Serial clock** : The serial clock is used to synchronize data movement both in and out of the device through its MOSI and MISO lines. The Master and Slave devices exchange 8 bits of information during a sequence of eight clock cycles. SCK is an output on the master device and an input on the slave device.
- **Master Out Slave In :** The MOSI line is configured as output in a master device and as an input in a slave device. It is one of the two lines that transfer serial data in one direction, with the most significant bit sent first.
- **Master In Slave Out :** The MISO line is configured as an input in a master device and as an output in a slave device. It is one of the two lines that transfer serial data in one direction, along with the most significant bit sent first.
- **Slave Select :** The slave select line is used to select one of the slave devices present. The slave select line is made low during data transmission and high when SPI is in idle state.



SPI Master and Slave

III. SPI DATA TRANSMISSION

The main element of the SPI system is the SPI Data Register. The 8-bit data register in the master and the 8-bit data register in the slave are linked by the MOSI and MISO pins to form a distributed 16-bit register. When a data transfer operation is performed, this 16-bit register is serially shifted eight bit positions by the serial clock from the master, so data is exchanged between the master and the slave. Data written to the master SPI Data Register becomes the output data for the slave, and data read from the master SPI Data Register after a transfer operation is the input data from the slave. The SPI system operates in two modes master and slave.



SPI Data Transmission

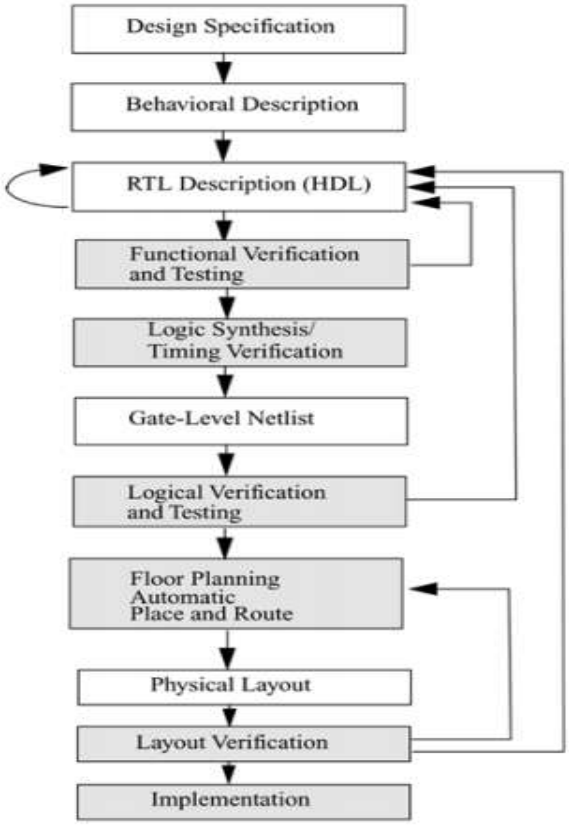
SPI MASTER MODE

The SPI is configured as a master when the MSTR bit is set. Firstly, the byte to be transferred to the slave is written to the SPI transmit data register. Then, the shift register is checked to see if it is empty. If it is empty, then, the byte is transferred from the data register to the shift register. After this transfer, another byte can be written to the data register if the SPTE control bit is set. Now, the SS bar pin is made low to the slave which we want to use. The master generates a clock that synchronizes the transfer of data from the master to the slave and vice versa. Then the byte begins shifting out a bit at a time on the MOSI pin synchronized with the master serial clock. The byte will be transferred for 8 clock cycles completing the transfer of data with the whole byte transmitted to the slave shift register. The data then transfers to the slave receive data register if it is empty. The SPRF control bit is set indicating that the SPI receive register is full and waiting to be read.

SPI SLAVE MODE

The SPI is configured as a slave when the MSTR bit is reset. When the slave needs to send a response back to the master, the master will continue to generate a prearranged number of clock cycles, and the slave will put the data onto MISO. The byte of data shifting a bit at time on the MISO pin is synchronized with the master serial clock. The master shift register is then automatically transferred to the master SPI receive data register if it is empty.

IV. FPGA DESIGN FLOW



FPGA Design Flow

Design Specification : This is the aspect of the prototype which is to be implemented on FPGA (Field Programmable Gate Array). There are various design constraints which the prototype should meet.

Behavioural Description : In this stage, the user relates the inputs and outputs in terms of functionality. It describes how the prototype will behave for a particular state of input conditions. It is basically a written logic between the inputs and outputs.

RTL Description : Here, we convert the behavioural description to an RTL description (Register Transfer Logic) in terms of hardware descriptive languages. RTL is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. RTL code is checked several times for accuracy.

Functional Verification and Testing : This is the stage where the RTL code is simulated and its functionality verified. If everything is verified and functioning properly, then, the user moves to the next stage. Else, the RTL description is to be modified and verified unless the functionality doesn't meet the behavioural description.

Logic Synthesis : Synthesis is the process of converting the RTL description into a set of logic gates and its interconnects. Conversion from RTL description to the gate-level netlist is achieved by logic synthesis tools.

Gate – Level Netlist : A gate-level netlist is a description of the circuit in terms of gates and connections between them. Logic synthesis tools ensure that the gate-level netlist meets timing, area, and power specifications.

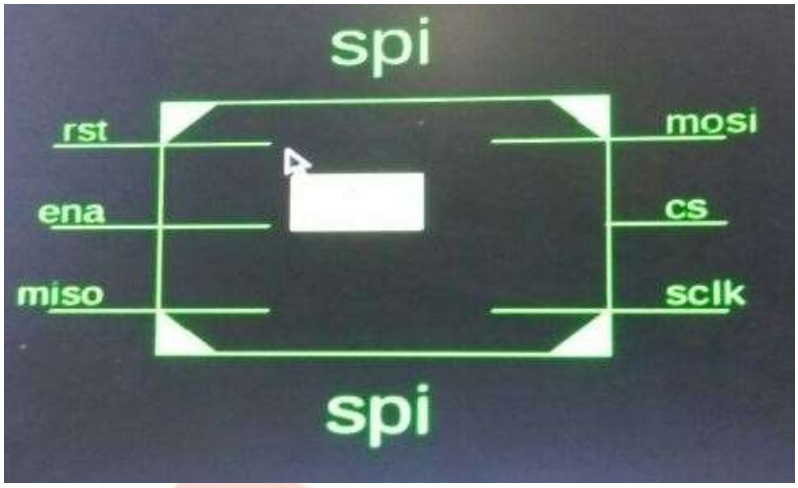
Placement : It is the placing of different logic gates, LUTs (Look Up Tables). In this design, placement is totally automatic. This stage creates a layout which is verified and then fabricated on the chip.

Routing : It is the providing of interconnections between different gates and logic units. Before routing, a user constraint file called UCF is created by providing the sides of clocks, input and output pins. These are written on the FPGA port. After placement and routing, the tool we use generates a bit stream file to be used in the next stage.

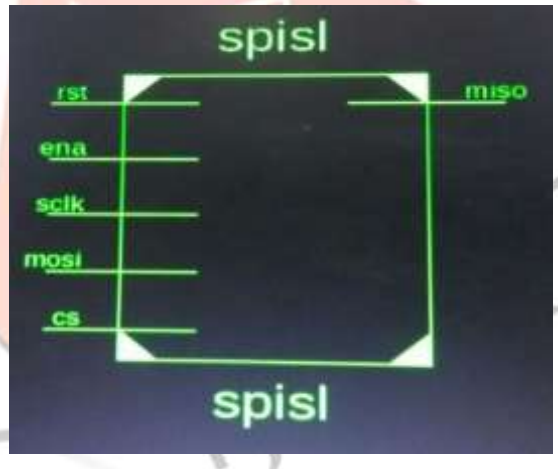
Finally, the FPGA is programmed by downloading the bit stream file. Then, the functionality of the FPGA is verified to check if it is meeting the design specifications.

V. IMPLEMENTATION AND SIMULATION RESULTS

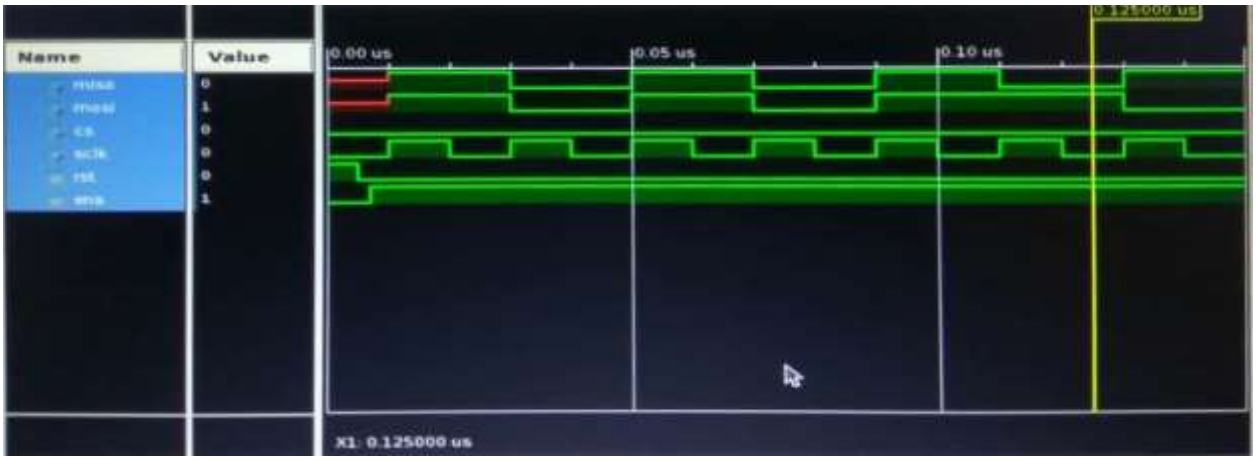
The whole design code, i.e. for synthesizing the SPI module and the verification of its functionality was accomplished in Verilog HDL (IEEE 1364 – 2001 compliant) using Xilinx ISE Design Suite 14.7.



RTL Schematic of SPI Master



RTL schematic of SPI Slave



Data Transmission from Master to Slave

VI. CONCLUSION

The SPI Master and Slave modules have been designed and implemented in Verilog HDL using FPGA Design Flow and Xilinx 14.7 was used for simulation. The main objective of this paper is to check the transmission of data from the master to the slave and verify the accuracy of the data transferred. The functionality of the module had also been verified and executed on Xilinx Virtex 5 FPGA Board.

REFERENCES

[1] Motorola Inc., "SPI Block Guide V03.06," February 2003
[2] Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition By Samir Palnitkar
[3] F. Leens, "An Introduction to I2C and SPI Protocols," IEEE Instrumentation & Measurement Magazine, pp. 8-13, February 2009.
[4] 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: 978-1-4244-50916
[5] L.Bacciarelli et al, "Design, Testing and Prototyping of a Software Programmable I2C/SPI IP on AMBA Bus," Conference on Ph.D. Research in Microelectronics and Electronics (PRIME'2006), pp. 373- 376, ISBN: 1-4244-0157-7, Ortanto, Italy, June 2006.