High Speed SPI Slave Implementation in FPGA using Verilog HDL

Mr. Akshay K. Shah

***Abstract*— SPI (Serial Peripheral Interface) is a synchronous serial communication interface for short distance communication. It is also called a four-wire serial bus. SPI Devices communicate in full duplex mode in Master-Slave architecture with a single master. It's operation is relatively very simple and operating speed is very high. The designed SPI Slave in FPGA will communicate with a DSP at relatively high speed.**

***Index Terms*— Serial Peripheral Interface (SPI), Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP). Printed Circuit Board(PCB), USB (Universal Serial Bus),**

I. INTRODUCTION

We generally prefer serial communication over parallel communication, as serial communication provides number of advantages like improved noise integrity, less number of pin counts and also high speed. We use different protocols for both long and short distance communication.

Long Distance: Ethernet, Serial-ATA (SATA), USB, etc.

Short Distance: I2C, SPI, etc.

- There are number of ICs of memories (i.e. FRAM, EEPROM, etc.) available in the market which, we can operate as a slave. We can readily use these ICs by implementing a SPI master implementation in either microcontroller/FPGA/DSP.
- But there are certain cases where we do not require that much memory in terms of number of registers as well as we want to manipulate that received data for other purposes in different application.
- In that cases we can build our user defined memory in FPGA and we can build a SPI Slave module inside FPGA that can interface serially with any SPI Master at very high speed in full duplex mode.

II. SPI PROTOCOL

SPI is a synchronous 4-wire protocol which uses 4-wires for establishing a full duplex communication between master and slave. As stated above, there can be only one master and any number of slaves can be connected.

Four Wire- Two control wires and two data wires:

Control Wires:

- SPICLK – Clock Signal generated by a master. This protocol is called synchronous because operation is controlled by this clock only.
- SPISTE (CS) – Chip Select. This signal is used by a master to select a slave from the number of slaves connected on the bus.

Data Wires:

- MISO/SOMI - Master In Slave Out. Data input from slave to master.
- MOSI/SIMO – Master Out Slave In. Data output from master to slave.

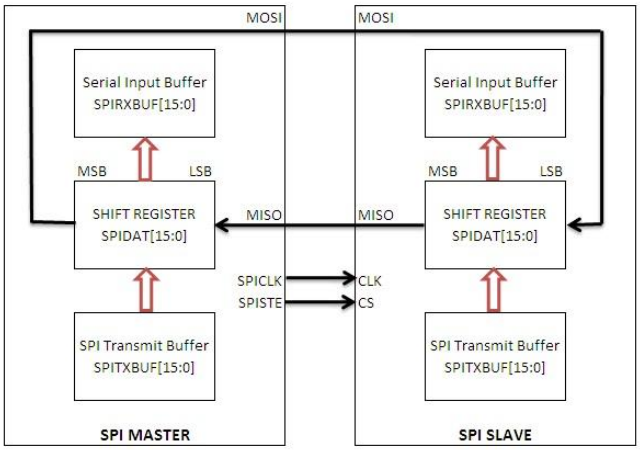

Fig.1 Data Transfer in SPI Interface

There are four operating modes available in SPI standard protocol which can be determined by the polarity of clock polarity (CPOL) and clock phase (CPHA). Both master and slave have to run in the same mode in order to achieve the proper communication between them.

SPI MODE	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

CPOL = 0:

CPHA = 0 – transmits data on rising edge and receives on falling edge of the SCLK signal (Rising edge without delay).

CPHA = 1 – transmits data one half-cycle ahead of rising edge and receives on rising edge of the SCLK signal (Rising edge with delay).

ISSN: 2278 – 1323

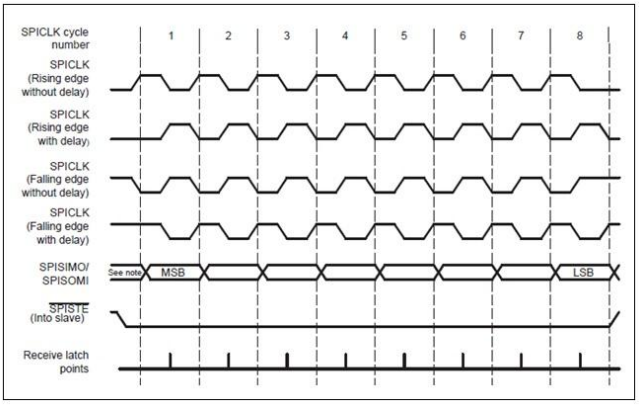

Fig.2 Different Modes of SPI Interface

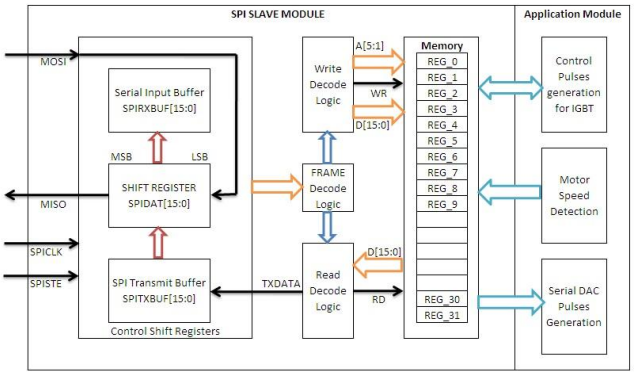

Fig.4 SPI Slave Module Implementation Block Diagram & Application Example

CPOL = 1:

CPHA = 0 – transmits data on falling edge and receives on rising edge of the SCLK signal (Falling edge without delay).

CPHA = 1 – transmits data one half-cycle ahead of falling edge and receives on falling edge of the SCLK signal (Falling edge with delay).

III. Proposed Implementation

SPI SLAVE MODE-0 (Rising edge without delay) configuration is implemented in FPGA. DSP acts as a master and slave module is implemented inside FPGA. FPGA global clock is driven by an external clock source.

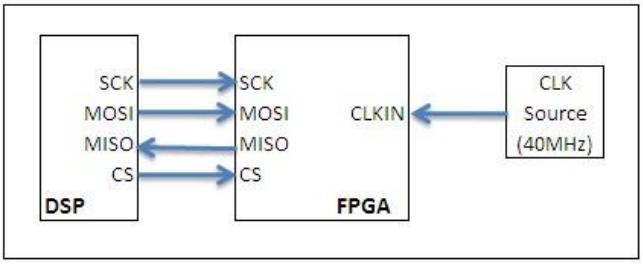

Fig.3 Block Diagram

A memory of 32 registers each of 16- bits wide is created inside FPGA which can be addressed with the help of 5 address lines. These address bits can be decoded from the incoming command bits.

As shown in Fig.4, there are different control logics inside slave module of FPGA, that can interpret the incoming frame from master and take decision according to the command present inside the frame.

Data written inside memory is utilized for number of applications as shown in Fig.4. The frame structure is as shown in the Table-1.

31---16	15---0
Command Frame	Data bits

Table-1 32-Bit Frame Structure

Command frame [16 bits] consists of synchronization bits (To check and confirm the clock & data synchronization), Read &/ Write command & 5 address bits while other bits are reserved for future use.

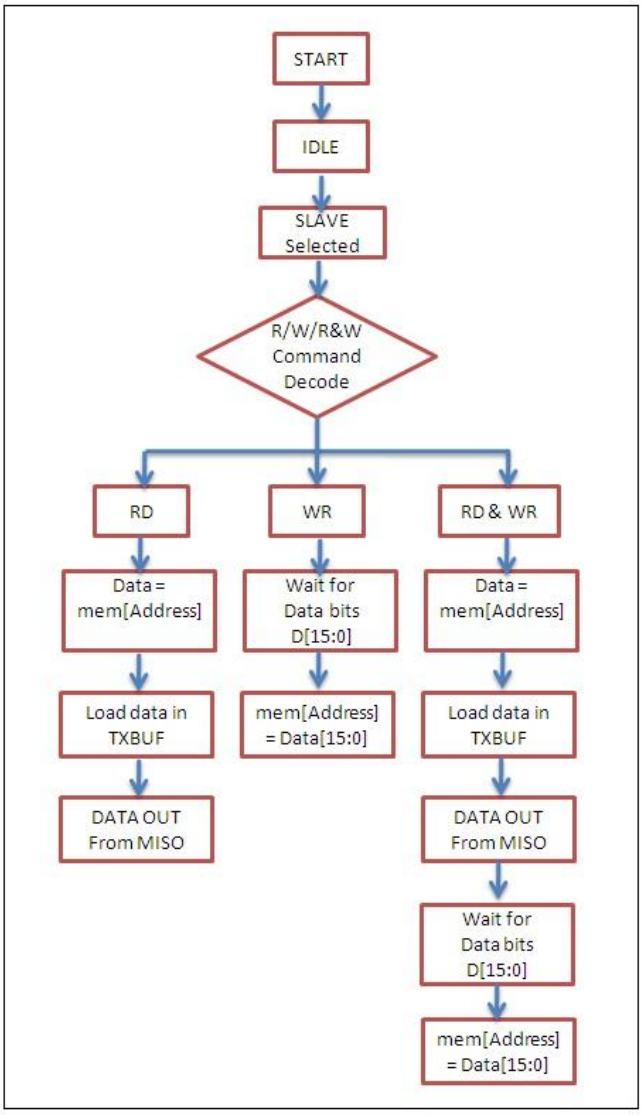

Fig.5 Decode Logic Flow Chart

Command frame coming from master decides the communication flow as under: [Fig.5].

ISSN: 2278 – 1323

- If WR (write) command is given from the master then Slave waits for the next 16-bits of data and after receiving all 32-bits, slave latched this data and writes data into the register of specified address.

- If RD (Read) command is given from master then, slave does not wait for the next 16-data bits. After receiving 16-bits of command, it loads the serial register with the data from register of specified address, and from the 17th clock slave transmits these data to the master on the MISO pin.

- If RD&WR (Both in a single command) are given then slave combines above two operations and operates in full duplex mode. After receiving first 16-bits it transmits the contents from specified address to master from 17th clock to 32nd clock. And after receiving complete frame it will write the new data bits into the same register of specified address.

IV. Implementation Results

SPI Slave is implemented and verified on ProAsic3 series FPGA (**P/N#** A3P600-PQ208).

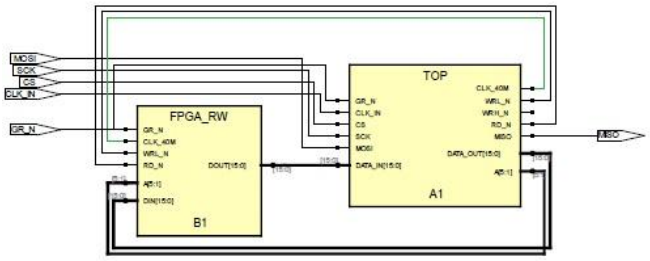

Fig.6 RTL View of the Implemented SPI Slave Module

Master SPI is implemented using DSP (**P/N#** F28M35H52C Concerto). FPGA code is written in Verilog HDL language and synthesized and compiled using Microsemi's Libero SoC v11.5 IDE.

Synthesis is done using Synplify Pro ME (version I-2014.03M-SP1).

Functional simulation is performed using ModelSim ME v10.3. Functional simulation results are displayed at 20 MHz SPICLK frequency in Fig.5. FPGA is programmed using FlashPro v11.5.0.26.

RTL view of the implemented block is shown in Fig.6. Implemented design has utilized 6 I/Os of FPGA. From Fig.6, we can see that design is partitioned into two separate blocks.

1) TOP (A1) – HIGH SPEED SPI SLAVE LOGIC
2) FPGA_RW – Memory of 32 Registers of 16-bits wide and addressing Logic.

Technology schematic diagram of implemented design is shown in Fig.7.

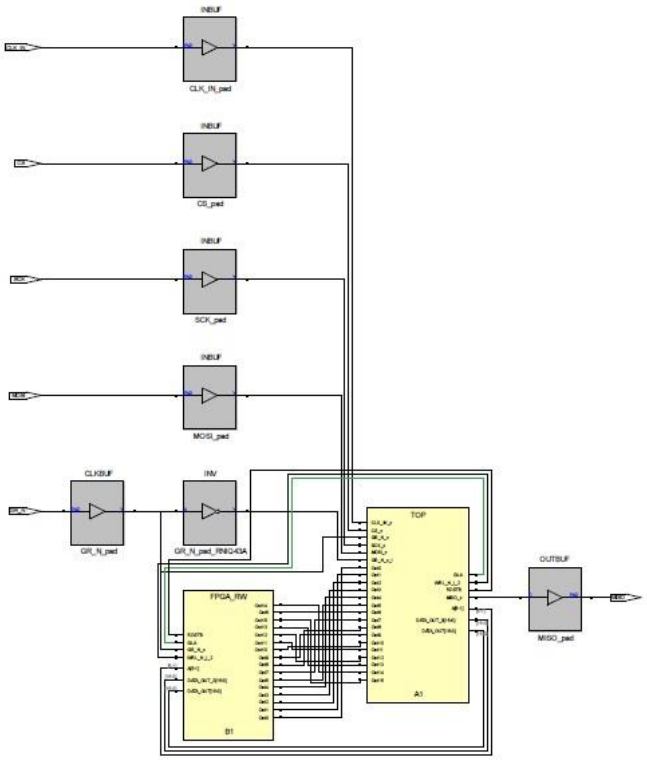

Fig.7 Technology Hierarchical View

After programming implemented design is verified completely. In Fig.8, overall design summary is given.

Run Status								
Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	31	0	0	-	0m:02s	-	12/16/2015 8:33:03 AM
Pre-mapping Detailed report	Complete	22	2	0	0m:00s	0m:00s	67MB	12/16/2015 8:33:05 AM
Map & Optimize Detailed report	Complete	8	4	0	0m:03s	0m:03s	75MB	12/16/2015 8:33:08 AM
Multi-srs Generator Detailed report	Complete				0m:00s			12/16/2015 8:33:04 AM

Area Summary			
Core Cells	1480	IO Cells	6
Block RAMs (v_ram)	0		
Detailed report			

Fig.8 Overall Design Summary

From area summary we can see that design has utilized 1480 core cells and six I/O cells. Resource utilization report (Detailed resource utilization report) is shown in Fig.9.

Functional simulation for all three commands is shown in Fig.10 and Fig.11.

From Fig.10, we can see that write command is given first from master to write 7878H data in REG3 from the 32 registers memory inside FPGA Slave. Then, a read command is given from master to read the contents of REG3. We can see that after getting command, Slave starts sending data of REG3 on MISO from 17th clock.

ISSN: 2278 – 1323

Target Part: A3P600_PQFP208_-2
Report for cell RW_SPI.verilog
Core Cell usage:

cell	count	area	count*area
AND2	9	1	9
AO1	171	1	171
AX1C	2	1	2
GND	122	0	0
INV	3	1	3
MX2	15	1	15
NOR2	8	1	8
NOR2A	16	1	16
NOR2B	358	1	358
NOR3A	1	1	1
NOR3B	22	1	22
NOR3C	57	1	57
OA1A	16	1	16
OR2	20	1	20
OR2B	3	1	3
OR3	153	1	153
PLL	1	0	0
PLLINT	1	0	0
VCC	122	0	0
XNOR2	1	1	1
XOR2	4	1	4
DFI1C0	2	1	2
DFN1C0	9	1	9
DFN1E0C0	1	1	1
DFN1E1C0	577	1	577
DFN1P0	12	1	12
DFN1P1C1	20	1	20
TOTAL	1726		1480

IO Cell usage:

cell	count
CLKBUF	1
INBUF	4
OUTBUF	1
TOTAL	6

Core Cells : 1480 of 13824 (11%)
IO Cells : 6

RAM/ROM Usage Summary
Block Rams : 0 of 24 (0%)

Fig.9 Resource Utilization Report

From Fig.11, we can see that master wants to read previous data of REG3 as well as write new data in a single command. From figure from 17th clock previous data is transmitted on MISO from slave and after receiving 32 bits new data is written in REG3 inside slave FPGA.

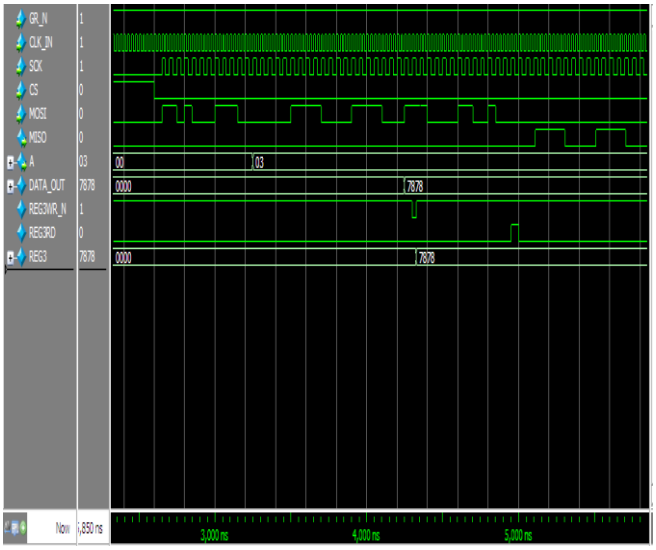

Fig.10 Simulation Results (Write Command & Read Command individually)

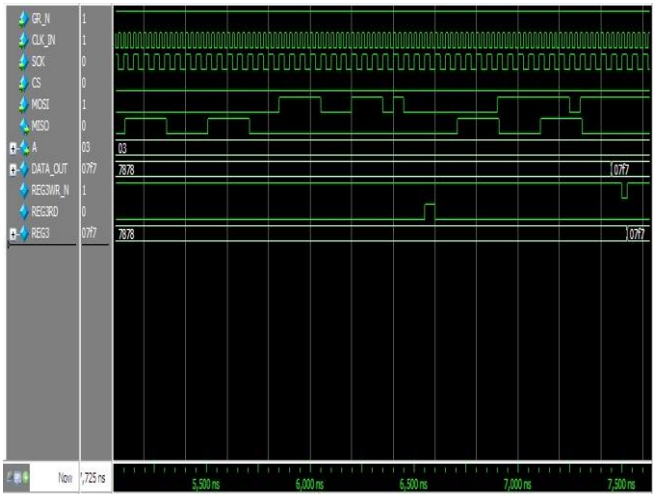

Fig.11 Simulation Results (Read and Write in a Single Command)

V. CONCLUSION

In this paper I have illustrated how to implement SPI Slave module in FPGA using Verilog HDL. The proposed design can be used with any SPI master device. This design is quite useful in the area where there is a requirement of high speed SPI interface. This design is written in Verilog HDL and fully verified by functional as well as timing simulation and through hardware implementation on the PCB.

REFERENCES

[1] Digital Logic Design By Morris Mano 2nd Edition.
[2] Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition By Samir Palnitkar. Publisher: Prentice Hall PTR. Pub Date: 21st February,2003. ISBN: 0-13-044911-3. Pages: 496.
[3] Contemporary Logic Design by Randy H. Katz, University of California, Benjamin Cummings/Addison Wesley Publishing Company, 1993.
[4] Proasic3 Flash Family FPGA Datasheet
[5] F28M35x Concerto Microcontrollers, Texas Instruments Datasheet

ISSN: 2278 – 1323

Mr. AKSHAY K. SHAH has received his B.E(Electronics & Communication) degree from Vishawakarma Government Engineering College, Chandkheda in June-2013 and currently working as an Engineer in Research & Development Department of Hitachi Hi-Rel Power Electronics Pvt. Ltd, since last two years. He has an interest in designing and verification of digital circuits using Verilog HDL, VHDL & System Verilog. He has hands on experience on working with CPLDs and FPGAs of Xilinks, Altera, Actel,etc.

ISSN: 2278 – 1323