Scientific Journal of Impact Factor (SJIF): 4.72

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 4, Issue 12, December -2017

DESIGN AND IMPLEMENTATION OF SPI PROTOCOL

Kamal Prakash Pandey, ApoorvUpadhyay

Department of Electronics and Communication, SIET, Allahabad

ABSTRACT :- *With advancements in VLSI, the devices can be configured for data transfer using the different types of data transfer protocols e.g. SPI (Serial Peripheral Interface), UART, Ethernet, I2C etc. This paper is about implementation of the low power and high speed Serial Peripheral Interface Protocol. In present we need to enhance the data transfer rate of devices to transfer data and consume less power so this paper mainly focus on speed and power consumption. SPI protocol is designed with Verilog HDL language, simulated and synthesized with XILINX VIVADO 2014.2 version software on zynq–7000 family board-xc7z020clg484-1.*

Keywords*: Data transfer protocols, SPI protocol, MASTER, SLAVE, Slave, MISO, MOSI, and SCK*

INTRODUCTION

The SPI (Serial peripheral interface) protocol has been already defined [1]. Serial peripheral interface protocol is used for low or medium speed intrachip and inter-chip data-stream transfers [2]. SPI is most commonly used protocol to communicate between different IC and peripheral devices. It is simple and has higher speed than other serial communication protocols like I2C, RS232 etc, it has been already implemented [6].SPI protocol has been designed and implemented over FPGA with BIST (Built In Self-Test) capability and it has been designed with Verilog HDL language and synthesized on Spartan 2 FPGA board [5].This paper focuses to implement efficient coding of SPI protocol to reduce its power consumption and increase speed. [2] A.K. Oudjida, M.L. Berrandji, R. Tiar, A. Liacha, K. Tahraoui have made a comparison between I2C and SPI in "FPGA Implementation of I2C & SPI Protocols: A Comparative Study" paper. This paper is a

comparison of physical implementation aspects of the two protocols I2C and SPI through a number of recent Xilinx's FPGA families. The RTL code is technology independent, inducing around 25% area overhead for I2C over SPI, and almost the same delays for both designs. [6]Pranjna, Mrs. Deepthi Dayanand, Shri Kanhu Charan Pandhy have worked on "VHDL Implementation of an SPI Interface for an FRAM Memory over FPGA". They have designed SPI which can be able to transfer data between spartan6 FPGA and FRAM. [7] Prakrity Porwal has researched on "Study and Implementation of SPI (Serial Peripheral Interface) using VHDL and its synthesis using Xilinx". This paper is mainly focused to generate the efficient coding for implementation of SPI Master Protocol. The RTL code is implemented in VHDL and the design code is simulated in Modelsim Se Plus 5.5c while its synthesis is verified using Xilinx ISE 14.1 version for Virtex 4 and Spartan 3 FPGA devices[4]. This paper has information about SPI and configuration registers of the bus guardian module of flexyRay controller.

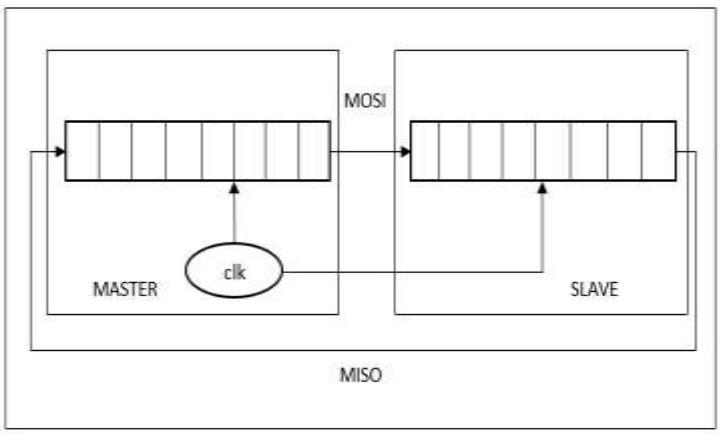


Fig.1 Data transfer in SPI [6]

Fig 1 represents the operation of shift register in SPI Protocol. MOSI is data bus by which master sends data to slave and by MISO, it receives data from slave device. Master device control the data transfer clock and selects slave device.CPHA=0, CPOL=0 – First bit starts as soon as SS is lowered, data sampled at rising edge, data changes on falling edge.

CPHA=0, CPOL=1 – First bit starts as soon as SS is lowered, data sampled at falling edge, data changes on rising edge.

CPHA=1, CPOL=0 – First bit starts on first clock edge, data sampled at falling edge, data changes on rising edge.

CPHA=1 , CPOL=1 – First bit starts on first clock edge , data sampled at rising edge , data changes on falling edge.

Functionality:

SPI works in two modes

1 – Master mode

2 – Slave mode

MASTER MODE – In master mode mosi, sck_t, ss (slave select), will act as an output whereas miso will act as an input. sck_t acts as clock for master which depends on global clock (clk).Master sends low signal on ss pin of particular slave to send data. Data transfer depends on two signals CPOL and CPHA, which define when the sck_t will latch the data and when data will be sample.

At CPOL=0 the base value of the clock is zero

CPHA=0, Data is captured on the rising edge of clock(i.e. low to high transition)

CPHA=1 data is captured on falling edge of clock and is propagated on rising edge

CPOL=1 the base value of clock is one

Control register first bit (control[1]) will decide either device will work in master mode or in slave mode.If it is one, then device will work as master and on zero work as slave.

CONTROL[0] – SPIE

SPIE will make enable to spi protocol

CONTROL[2] – CPOL

CONTROL[3] – CPHA

CONTROL[4] – MSB/LSB

If it is '0,' then SPI first send MSB (most significant bit) and if '1,' then LSB (least significant bit).

CONTROL[7:5] – R Reserved

CONTROL RESISTER –CONTROL[7:0]

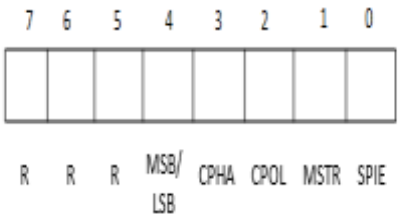


Fig. 2 control register

SPI protocol uses two data lines

1. MOSI – Master Out Slave In
2. MISO – Master In Slave Out

MOSI – Master Out Slave In

SPI sends data through MOSI data line to slave device

MISO – Master In Slave Out

SPI protocol receives data from slave through MISO data line

SS – Slave Select

Master selects the slave device by sending low signal on this pin which is connected to slave .This line has active low signal. Master can send data to many numbers of Slave devices but only one at a time

sck_t - sck_t is a clock at which device works as a master or slave. At one edge of clock, (positive or negative) data is sampled and on other edge, data transfer depends on value of CPOL and CPHA.

SPIE – It is a '0'th bit of control resister, which enables SPI.
SPIE =1, SPI is in enable state
SPIE =0, SPI is in disable state
MSTR – It is 1st bit of control resister (CONTROL[1])
MSTR = 0, SPI will work in master mode
MSTR = 0, SPI will work in slave mode
MSB/LSB – It is 4th bit of control register (CONTROL [4])
MSB/LSB = 0 First LSB bit of shift resister will be sent
MSB/LSB = 1 First MSB bit of shift resister will be sent

STATUS REGISTER

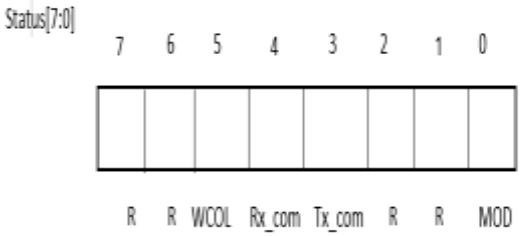


Fig.3 status register format

R – Reserved bit
WCOL – Collision bit [5]
This bit will be one when transmission occurs to indicate, you will have to wait until transmission is completed otherwise collision of data will happen and data will be lost.
Rx_com –receiving complete
= 1 receiving of 8-bit data complete
= 0 receiving of 8 bit data in progress
Tx_com – transmission complete
=1 transmission of 8 bit data complete
=0 transmission of 8 bit data in progress
MOD – operating mode
=1 master mode
=0 slave mode

FLAG RESISTERS

txflag – Transmission flag
If txflag is '1' then transmission and receiving will occur if '1,' then transmission and receiving will stop.
Prescalerflag – This flag register is used to control frequency division.If this flag is '0' then preclear register will pick the value from max register and divide frequency accordingly.
Width – width is used as parameter to give the values to shift register for shifting.
mosi_t and miso_t – These registers are used to store shifted bit MSB or LSB of shift register.
sck_temp – This is one-bitregister, which is used to select initial value of clock sck_t.
Slave select – This is 32 bit register for selection of slave device.
Counter – It is 4 bit register to count clock sck_t for 8 bit data transfer and after 8 bit data transfer its value becomes '0'.
ss – It is 8-bit register to select slave device .It works as active low i.e. slave is activated when particular bit becomes low.
reg clk – It is used to generate global clock .
sck_t – sck_t register is used to give clock to protocol.
Ssbar – controls the SPI protocol in slave mode
=0, works in slave mode
Strobe - controls the SPI protocol in master mode
=1, work in master mode

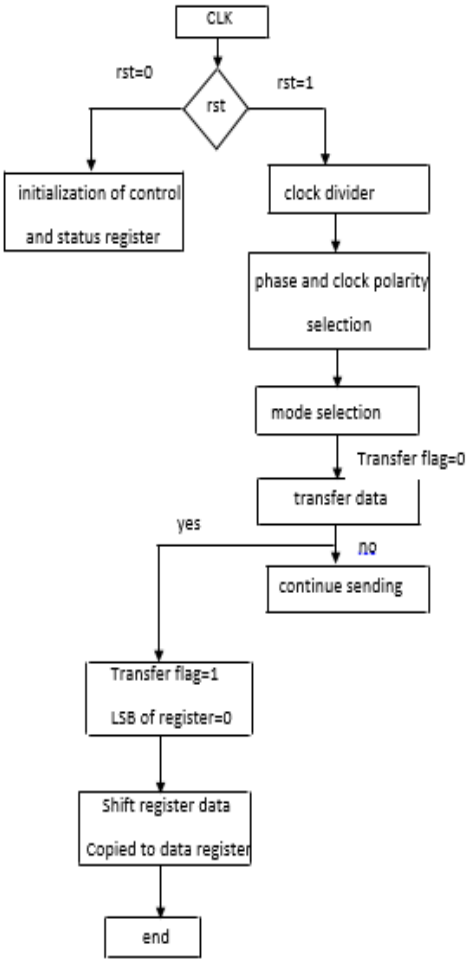


Fig.4 master mode flow chart

Flowchartdescription of data flow in master mode-

In master mode, system generates clock. At the positive edge of clk(clock) if rst(reset) is made zero, then all registers get reset after that if rst and strobe signal become one and if control[3] bit is one, then data from data_xmit register is copied into shift register in shifted form or same as data_xmit register ifcontrol[3] is zero and after that at the negative edge of clock prescalar, block divides the clock according to prescalar value then control[3:2] bit decides the clock polarity and phase of clock sck_t and counter starts from there and if control[0] bit is '1,' then it will work in master mode otherwise in slave mode.Transmission flag tx_flag gets '1' during 8 bit, data transfer and after that it becomes '0'and received data copied to data receive register (data_rcv) .After 8 bit of data transfer a new value is loaded into shift register from data transmit register.

SLAVE MODE – The SPI works in slave mode when the MSTR bit in control resister is clear. In slave, mode clk is the clock input from master, and ss is slave select input. Before a data transmission occurs, the ss pin of slave SPI must be at logic low and ss should be low until transmission is complete.

Baud rate generation – SPI can work on different frequencies. It can be done by assigning value in prescaler reg which will divide the clock frequency in desired one.

e.g.prescaler = 8'b00000010 fclk=fclk/2
prescaler = 8'b00000100 fclk=fclk/4
prescaler = 8'b00001000 fclk=fclk/8

Different baud rates are required because different peripheral devices work on different speeds. So communicate to those devices, we have to transfer data according their speeds. If baud rates are not matched then the data will be lost and the value received by peripheral device will be garbage.

In slave mode if ss pin is low, the first bit in data register is driven out through data out pin.If slave is not selected, then the clk input is ignored and no data will be transferred.

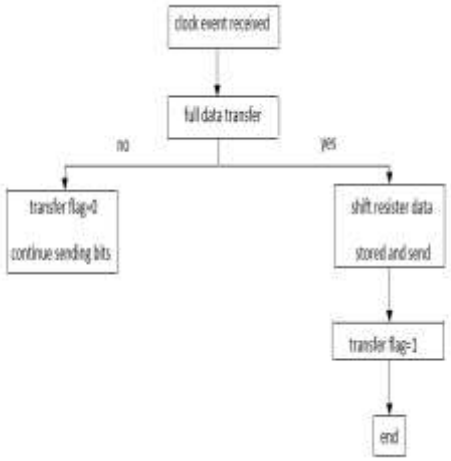


Fig.5Slave mode flow chart

Flow chart description data flow in slave mode

Slave mode is selected by giving control [1]bit '1' and ssbar '0'.When clock received by master device then transfer flag gets value '0' and ss pin gets '0,' then starts transfer and receives 8 bit data after 8 clock pulses transfer flag gets '1,' and received data is stored from shift register to data receive register (data_rcv).In slave mode clock polarity and phase and ss in logic is controlled by master device.

Different baud rates are required because different peripheral devices work on different speeds. So communicate to those devices we have to transfer data according to their speeds. If baud rates are not matched, then the data will lost and the value received by peripheral device will be garbage.

IMPLEMENTATION AND SIMULATION RESULT

To initialize data transfer between master and slave control register should be programmed. By Selecting, appropriate bit value of control register SPI can be operated in master mode or slave mode. The status register gives information about status of data transfer, whether the data has been transferred completely or not .The design of SPI module is done in Verilog HDL and verified and synthesized using Vivado 2014.2 .Below fig depicts the simulation result showing the SPI interface signals from the master and slave. The design mapped on zynq–7000 family board-xc7z020clg484-1 device. This device can work up to n*10MHz speed.Waveform for master for data_xmit (data to be transmitted to the slave) =00000001

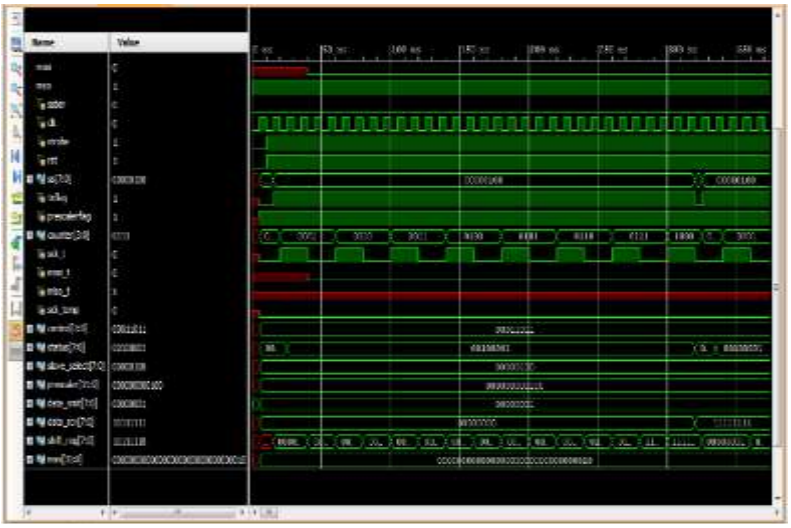


Fig.6

Fig.6 shows, the master device have successfully transferred the data from data_xmit register and received data from slave on data_rcv register.Here the value of control register is"00011011".

Waveform for slave for data_rcv (data received from master) =11111111

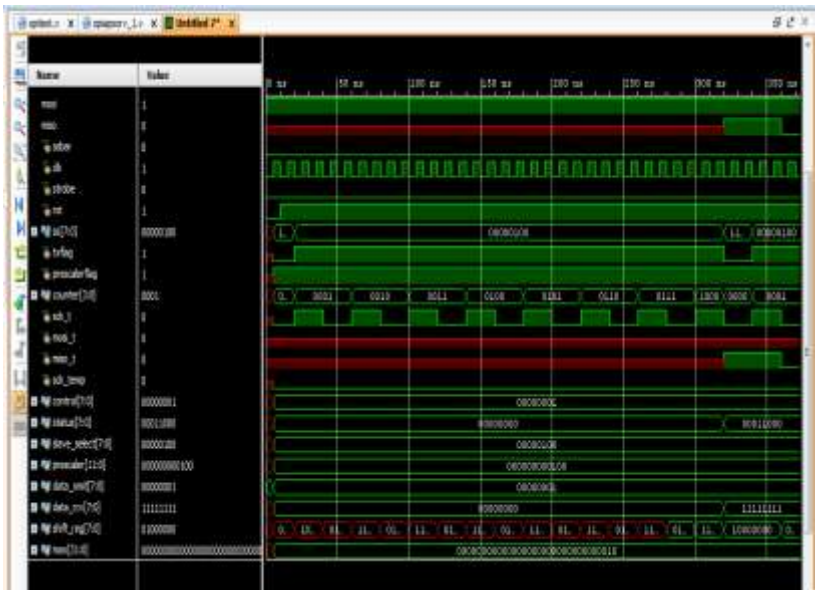
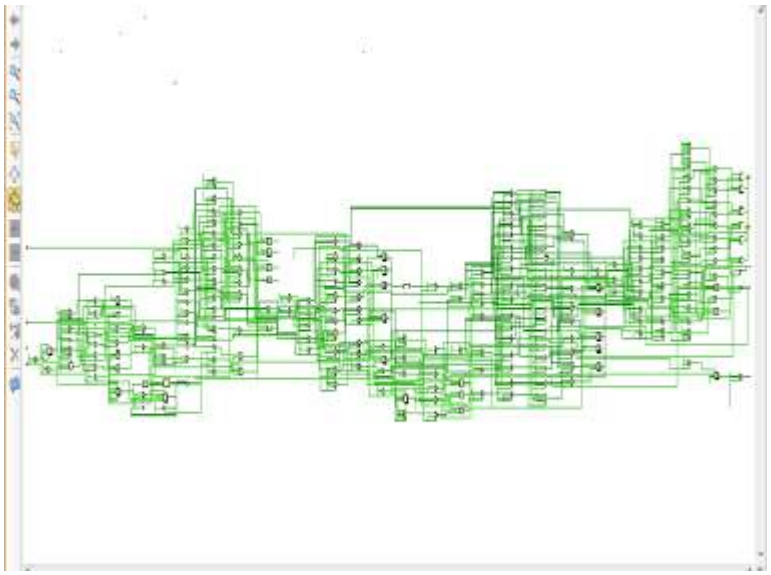


Fig.7

Fig.7 shows the slave device have, successfully received, the data from data_rcv register and transferred data to master from data_xmit register. Here the value of control register is"0000001".

RTL Schematic of SPI Protocol design



Synthesis report

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	22	0	53200	0.04
LUT as Logic	22	0	53200	0.04
LUT as Memory	0	0	17400	0.00
Slice Registers	70	0	106400	0.06
Register as Flip Flop	70	0	106400	0.06
Register as Latch	0	0	106400	0.00
Bonded IOB	13	0	200	6.50

Total On-Chip Power (W)	0.123
Dynamic (W)	0.003
Device Static (W)	0.120
Effective TJA (C/W)	11.5
Max Ambient (C)	83.6

CONCLUSION

This SPI is much more flexible, has improved speed and cost.It consumes less power with respect to conventional one.This work is focused on implementation of SPI protocol for both master and slave mode. The entire design serial peripheral Interface SPI protocol is designed with Verilog HDL language,simulated and synthesized with XILINX VIVADO 2014.2 version software on zynq–7000 family board-xc7z020clg484-1. This design can be used in a system, which requires low power and high-speed data transfer. It can connect many no of slave devices but one at a time and can be used for different slow and medium speed peripheral devices to data transfer.

REFERENCES

[1] F. Leens, "An introduction to I2C and SPI protocols," IEEE Instrumentation & Measurement Magazine, vol.12, no.1, pp.8-13, February 2009.

[2] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, K. Tahraoui, "FPGA implementation of I2C & SPI protocols: A comparative study,"in Proc. 16th IEEE International Conference on Electronics, Circuits, and Systems, pp.507- 510, Dec. 2009.

[3] N.Q. B. M. Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," in Proc. 2012 IEEE Symposium on Industrial Electronics and Applications (ISIEA), pp.78-83, Sept. 2012.

[4] A.N. Gaidhane and M.P. Khorgade, "FPGA Implementation of Serial Peripheral Interface of FlexRay Controller," in Proc. 13th International Conference on Computer Modelling and Simulation (UKSim), pp.128-132, Apr. 2011.

[5] Shumit Saha, Md. Ashikur Rahman, Amit Thakur, "Design and Implementation of SPI Bus Protocol with Built-In-Self-Test Capability over FPGA", in International Conference on Electrical Engineering and Information & Communication Technology (ICEEICT) 2014.

[6]Prajna, Mrs. Deeepthi Dayanand , Shri Kanhu Charan Padhy, "VHDL Implementation of an SPI Interface for an FRAM Memory over FPGA", in International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 4 Issue 4, April 2015.

[7]Prakriti Porwal, "Study and Implementation of SPI (Serial Peripheral Interface) using VHDL and its synthesis using Xilinx",in International Journal of Innovative Research in Science, Engineering and Technology Vol. 5, Issue 6, June 2016.

