International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 11, November - 2013

FPGA Implementation Of SPI To I2C Bridge

Abhilash S.Warrier
VLSI Design
VNIT Nagpur

Akshay S.Belvadi
VLSI Design
VNIT Nagpur

Dhiraj R.Gawhane
VLSI Design
VNIT Nagpur

Babu Ravi Teja K
VLSI Design
VNIT Nagpur

Abstract

Today's electronic system is not a standalone unit instead working in a group, where each IC communicates with each other very frequently, it is necessary to have an efficient simple protocol for effective communication among these components, the major area being serial communication. SPI and I2C are two widely accepted and practised global standards for both inter-chip and intra-chip serial communication for low/medium bandwidth. The paper discusses the two protocols in detail and a SPI to I2C Bridge. Design and FPGA implementation of this bridge is in conformity with design reuse methodology.

Key words- *I2C, SPI, SPI to I2C bridge, FPGA.*

1. Introduction

For low end low/medium bandwidth serial communication the two worldwide accepted standards are SPI and I2C. The two protocols have their own prospects and are strong competitors to each other. A comparative study of the two protocols is discussed in [3]. By and large there is a high probability that they coexist in the digital world. With a lot of peripheral devices from a variety of vendors there is always a choice to be made between the two. The major emphasis of the paper is to implement a SPI to I2C Bridge through which there can be a seamless communication between the I2C SLAVE and the SPI MASTER. However, there is a large area overhead involved in this process. The paper initially discusses the major aspects about the I2C and SPI protocols. The detailed explanation of SPI and I2C and their specifications can be found in [7] and [6] respectively. The later part explains the SPI to I2C Bridge modelled as a FSM.

2. Inter Integrated Circuit (I2C)

I2C is a serial, short range low level communication protocol with multi master capability. It is the simplest of its kind with only two external world connections viz. SDA (Serial Data) and SCL (Serial Clock) line.
The three modes of operation of I2C are:

a. **Standard mode** - data transfer up to 100 Kbps
b. **Fast mode** - data transfer up to 400 Kbps
c. **High Speed mode** - data transfer up to 3.4 Mbps.

The maximum device driven capability is limited just by the total drivable load capacitance which should not exceed 400pF.

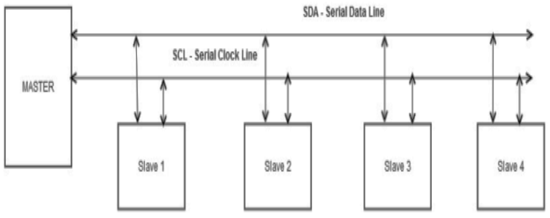


Fig.1. I2C block diagram

Data transfer takes place via the SDA and SCL lines. Any data transition can occur on SDA line only when the SCL line is on Logic Low and all these transitions have to settle down before the SCL goes to Logic high. I2C is a bit oriented bi-directional protocol there by facilitating a Serial Full Duplex communication between the master and the slave.

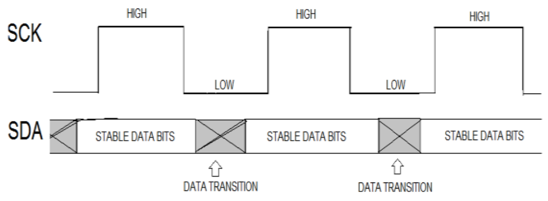


Fig.2 DATA transfer using I2C

Since, it is a bit-oriented protocol there are special conditions defined for the START and STOP conditions of the data transmitted on SDA line. START is identified when there is a falling edge (HIGH to LOW) detected on SDA line when SCL is in Logic HIGH state. STOP is identified when there is a rising edge (LOW to HIGH) detected on the SDA line when SCL is in logic HIGH state.

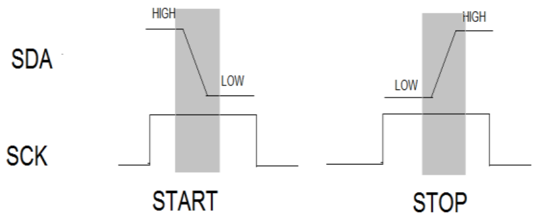


Fig.3. START and STOP conditions in I2C

There is another special condition in which a START condition is repeated instead of a STOP following a START and it is treated as a Repeated START case which is logically equal to a fresh data transfer state. At any given point of time the I2C bus can be in any of the two states **BUSY**: A state after START and before STOP. **IDLE**: A state after STOP and before the next START.

MASTER is any device that which has control over the SCL line and has the capability to initiate and terminate a data transfer, it can also control the addressing of other devices connected as slaves. SLAVE is just any device which is just capable of sending or receiving data and addressed by a MASTER. I2C allows the master both to transmit as well as receive data to or from a slave. Multi MASTER Mode is also allowed. In such a case an ARBITRATION process takes place to decide which master is going to control the bus.

2.1 Data transfer on I2C

Data Transfer is Byte oriented and the number of data Bytes transmitted per transfer is unrestricted. Data bits are transferred one bit per clock cycle after the START condition. Byte is a combination of eight data bits and an Acknowledgement bit is necessary after every byte. Transmission takes place as MSBF (Most Significant Bit First). Initially MASTER sends the Address of this slave with which it wants to initiate a communication and the further communication depends on the Acknowledgment bit from SLAVE.

2.2 Acknowledgement

When the Master has completed 8 data bits transmission, it will let the SDA line be in logic HIGH state. During the next clock pulse if the SLAVE pulls down the SDA line when the SCL is high, then it is an Acknowledgement received from the slave that the data has been received. The Slave can now Hold the SCL Line in Logic Low state which will push the master into Wait state else the master will continue with the next set of data. To give a NACK (Negative Acknowledgement) the slave will simply let the SDA line to remain in its logic HIGH state during the ninth clock pulse. The Master in such a case will generate either a STOP or a Repeated Start condition. In case, if the MASTER is acting as a receiver for a slave transmitter, the MASTER signals the end of data transmission by not acknowledging the last Byte of the data stream. The slave releases the SDA line and MASTER can now generate a STOP or Repeated Start condition.

2.3 Arbitration

I2C being Multi-Master capable, has a well-defined Arbitration process. Any Master which wants to transmit data will search for the availability of a free SDA line, if one or more masters have data to be sent. They generate the START bit at the same instant and now there is a resource conflict. The master which is trying to drive the SDA line will sample the value of SDA and compares it with what it has to send. If they match it continues to transmit, else it has lost the arbitration process for that Byte transfer period and may generate clock pulses on SCL till the end of that Byte period. If the losing master has a slave mode too then it immediately switches to its slave mode. Since, the first ever transmission is the address of the slave with which the connection is to be established and both the masters want to communicate with the same Slave, then the arbitration continues. Arbitration takes place on the data bits if the master is the Transmitter or on Acknowledge bits if the master is working as a receiver. Since the data on the SDA line is the same as that of the winning MASTER, ideally no data is lost during the Arbitration process. Arbitration is only for masters and slaves can’t take part in it.

2.4 Synchronization

All the Masters can generate its own clock but for a bit by bit arbitration process it is necessary to have a well-defined clock. For all the devices that are willing to communicate via the SDA of I2C, their clock signals are wired together and the SCL will be in LOW state for duration equal to the Maximum Clock LOW period and will be in logic HIGH for a duration which is the minimum Clock HIGH.

2.5 Frame format

The I2C connected devices are capable of two way communication with their slave. The MASTER can either act as a transmitter or a receiver, in the sense that it can either write data to a slave or read data from the slave. Every slave connected to the bus has a unique address in a 7 bit address format. The eighth bit of the first transmitted byte will tell whether the MASTER is going to work as a Transmitter or a Receiver. If eighth bit is a zero it says that MASTER is transmitter else it’s a receiver.

MASTER acting as transmitter

S	Slave address	0	A	DATA	A	DATA	N	P

MASTER acting as Receiver

S	Slave address	1	A	DATA	A	DATA	N	P

Fig.4. Frame formats of I2C Master acting as transmitter and receiver

S- Start indication
P- Stop condition
A- Acknowledgement
N – Negative Acknowledgement

3. Serial Peripheral Interface (SPI)

SPI facilitates a Synchronous, Duplex and Serial Communication between the Peripherals. It supports up to 400 Mbps Data transfer speed between various peripherals and the Microprocessor. Similar to its counterpart SPI also provides a simple interface with only four pins to the external world MISO, MOSI, SCLK and SSN.

3.1 Pin description

SCLK: Serial Clock line used to synchronize the data transfer. Master only can generate the Clock.
MISO: Master In Slave Out also referred as Serial Data Input (SDI) line used to send data to the master from a slave.
MOSI: Master Out Slave In also referred as Serial Data out (SDO) line used to send the data from master to slave.
SSN: Slave Select signal which is active low line and for each device connected on the bus the master dedicates a special line for each slave entity. So to drive 'n' no. of slaves on the bus master should have 'n' SSN lines.

3.2 SPI registers

There are various internal registers in the SPI they are as follows in the increasing order of their addresses: SPI Control Register 1, SPI Control Register 2, SPI Baud Rate Register, SPI Status Register and SPI Data Register. The Detailed Description of these registers is discussed elsewhere [7].

3.3 Data transfer

Data Transfer between MASTER and SLAVE takes place with the help of a SPI Data Register. This Register is connected as a shift register and is connected to MOSI pin in MASTER and MISO pin in the slave. The data is clocked in and out of the register in FIFO format. When a Device is not selected it has to Tri-state the MISO line. There is an option of multiple receivers by Buffering but multiple transmitters is not allowed because there will be contention problem on the Bus.

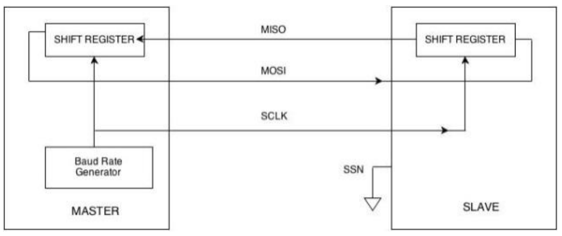


Fig.5. SPI data transfer block diagram

3.4 Synchronization

During SPI transmission Data is handled serially at both the master and slave sides. The SCLK line supervises the data transfer on both the data lines. SPI Control register 1 has two bits CPOL and CPHA which will decide whether the clock signal is an active HIGH or Active LOW signal and on which edge the data transfer takes place. CPHA = '0' indicates the data is to be sampled on Leading edge and CPHA='1' Indicates Data is to be sampled on trailing edge. Both the MATER and SLAVE are to be initialized with the same values for communication.

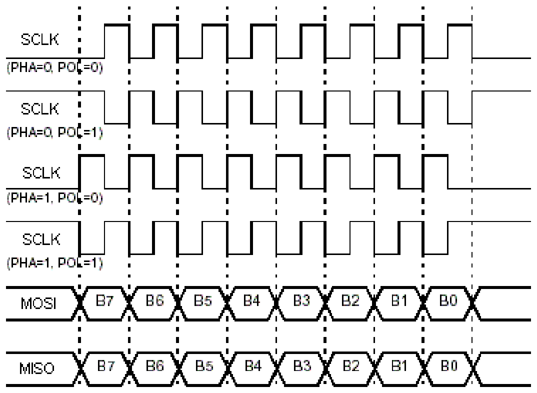


Fig.6. Synchronization and data transfer in SPI

4. SPI to I2C Bridge

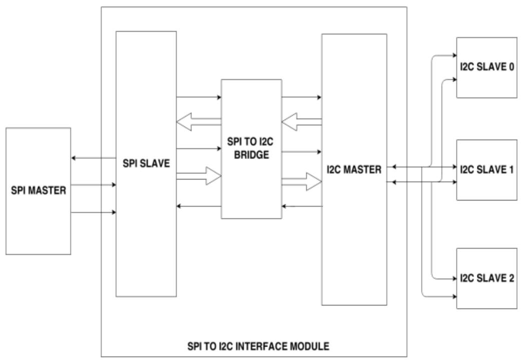


Fig.7 Block diagram of SPI to I2C Bridge

The SPI MASTER which wants to drive the I2C slave does so via the interface module. It includes a SPI SLAVE directly driven by its MASTER and our SPI to I2C Bridge interfaces SPI SLAVE and the I2C MASTER which in turn drives its SLAVE as shown in fig.7. The designed SPI to I2C Bridge acts a converter and bridges the SPI slave to an I2C master. The state diagram of the device is shown in fig. 8.

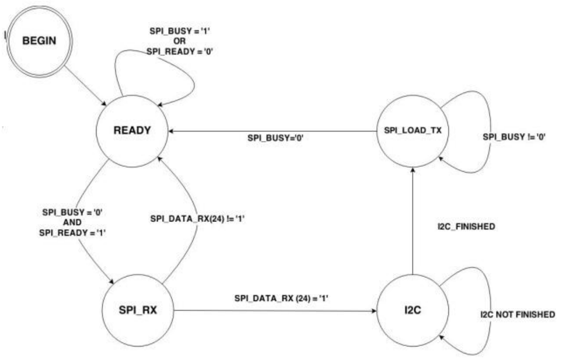


Fig.8. State diagram of the SPI to I2C Bridge

The SPI to I2C Bridge waits in READY state and continues to be in that state as long as spi_busy is HIGH or spi_ready is LOW. If there is an event which makes spi_busy LOW (saying that the SPI slave is not busy) and spi_ready HIGH (which indicates a new data arrival from the SPI master) then it receives data from the SPI Slave thereby entering into SPI_RX state. Then SPI data is latched. Then it checks for a I2C transaction enable bit which is 25th bit of received data and if it is HIGH it proceeds to the next state the I2C state else it will get back to the READY state .When in I2C state it waits for the I2C data transfer to be finished. If the SPI is not busy then it again moves to the ready state after writing the I2C transaction to the slave-transmission-register.

5. Results and discussion

The designed SPI to I2C Bridge has been coded using VHDL and simulated using Model-sim SE 6.2b. Synthesis is done using Xilinx ISE Design suite 14.2 and implemented on Spartan 3E FPGA.

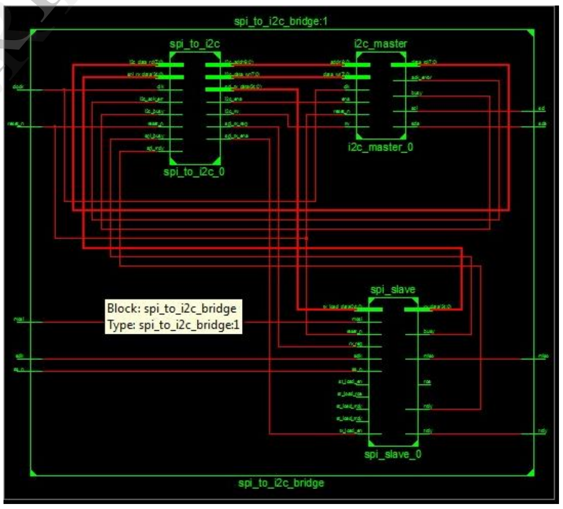
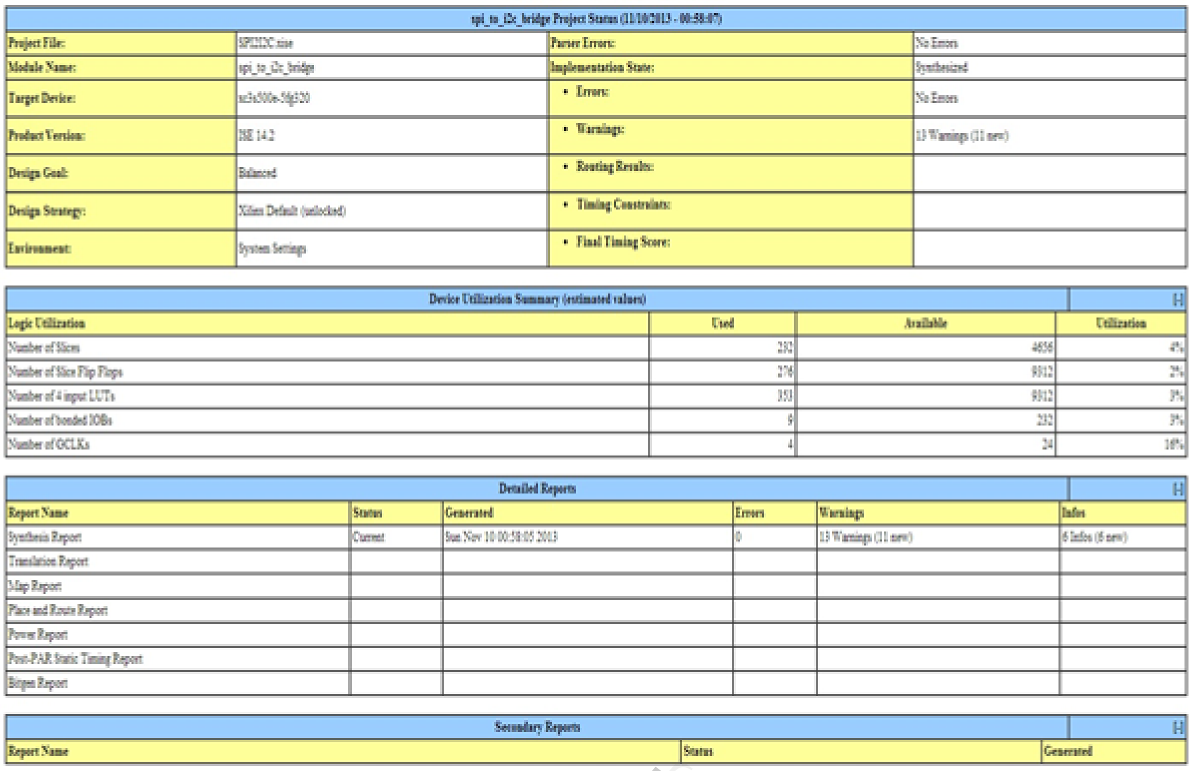


Fig.9. RTL schematic of the SPI to I2C interface module

6. Conclusions and future scope

The paper has shown FPGA implementation of SPI to I2C Bridge. This enables I2C slaves to be driven by SPI master. This will introduce a large area over head with the bridge consuming more area than the I2C and SPI itself .So power area optimization of this bridge is a challenge for the designers.



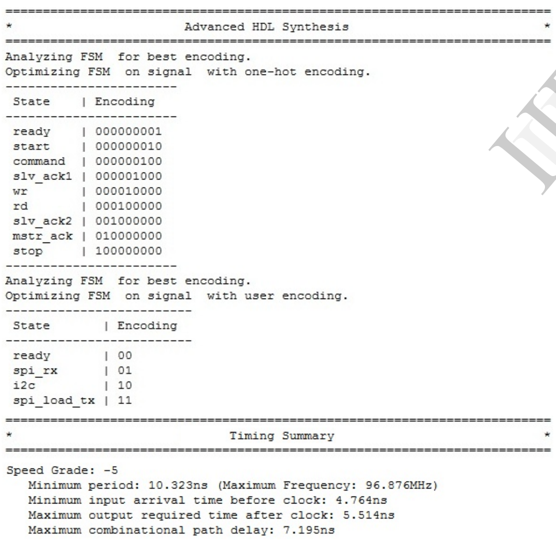
spi_to_i2c_bridge Project Status (11/10/2013 - 00:58:07)			
Project File:	SPI2I2C.xise	Parser Errors:	No Errors
Module Name:	spi_to_i2c_bridge	Implementation State:	Synthesized
Target Device:	xc3s500e-5fg320	• Errors:	No Errors
Product Version:	ISE 14.2	• Warnings:	13 Warnings (11 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	232	4656	4%
Number of Slice Flip Flops	276	9312	2%
Number of 4 input LUTs	353	9312	3%
Number of bonded IOBs	9	232	3%
Number of GCLKs	4	24	16%

Detailed Reports					[-]
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sun Nov 10 00:58:05 2013	0	13 Warnings (11 new)	6 Infos (6 new)
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports		[-]
Report Name	Status	Generated

Fig.10. Design summary for SPI to I2C Bridge generated using Xilinx ISE



```
=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================
Analyzing FSM  for best encoding.
Optimizing FSM  on signal  with one-hot encoding.
----------------------
 State    | Encoding
----------------------
 ready    | 000000001
 start    | 000000010
 command  | 000000100
 slv_ack1 | 000001000
 wr       | 000010000
 rd       | 000100000
 slv_ack2 | 001000000
 mstr_ack | 010000000
 stop     | 100000000
----------------------
Analyzing FSM  for best encoding.
Optimizing FSM  on signal  with user encoding.
-------------------------
 State       | Encoding
-------------------------
 ready       | 00
 spi_rx      | 01
 i2c         | 10
 spi_load_tx | 11
=========================================================================
*                            Timing Summary                             *
=========================================================================

Speed Grade: -5
   Minimum period: 10.323ns (Maximum Frequency: 96.876MHz)
   Minimum input arrival time before clock: 4.764ns
   Maximum output required time after clock: 5.514ns
   Maximum combinational path delay: 7.195ns
```

Fig.11. Synthesis report – FSM and Timing summary

Acknowledgement

The Authors would like to thank the Department of Electronics Engineering, Visvesvaraya National Institute of Technology, Nagpur for Technology Support.

References

[1] Prof. Jai Karan Singh et. al. Design and implementation of I2C master controller on FPGA using VHDL International Journal of Engineering and Technology (IJET), Aug-Sep 2012 , 4(4),162-166.

[2] Arvind Sahu et.al. An Implementation of I2C using VHDL for DATA surveillance. International Journal on Computer Science and Engineering (IJCSE), May 2011,3(5),1857-1865.

[3] A.K. Oudjida, M.L. Berrandjia, R. Tiar, A.Liacha, K. Tahraoui, "FPGA Implementation of I2c & SPI Protocols: A Comparative Study" Electronics, Circuits, and Systems, 2009. ICECS 2009

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

[5] J.M. Irazabel & S. Blozis, Philips Semiconductors, "I2C-Manual,"Application Note, ref. AN10216-0, March 2003.

[6] Philips Semiconductors, "The IIC-Bus Specifications," version 2.1, January 2000.

[7] Motorola Inc., SPI Block Guide V03.06, Feb. 2003.