Design and implementation of FPGA based interface model for scale-free network using I2C bus protocol on Quartus II 6.0

P. Venkateswaran
Jadavpur University
123 PUBLICATIONS **736** CITATIONS
SEE PROFILE

Madhumita Mukherjee
Jadavpur University
8 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Arindam Sanyal
University at Buffalo, State University of New York
115 PUBLICATIONS **1,723** CITATIONS
SEE PROFILE

Rn Nandi
Jadavpur University
86 PUBLICATIONS **619** CITATIONS
SEE PROFILE

Design and Implementation of FPGA Based Interface Model for Scale-Free Network using I2C Bus Protocol on Quartus II 6.0

P.Venkateswaran, Madhumita Mukherjee, Arindam Sanyal, Snehasish Das and R.Nandi

***Abstract* — To enable devices to communicate with each other over a serial data bus without data loss, as well as to enable faster devices to communicate with slower ones, the I^{2}C(Inter IC) protocol was put forward by Philips Semiconductors in January 2000. Ever since, there has been families of I^{2}C enabled microcontrollers like the PIC18F452 from Atmel and TMS470 from Texas Instruments. However, configuring these microcontrollers requires a lot of programming and knowledge of the register structures, and hence they are not portable. In this paper, a generic design on an FPGA platform is presented, which does away with the need of any further programming while setting up the network. Also, the proposed model can be used both as a master and as a slave. The entire design has been coded in VHDL and verified using Quartus II 6.0.**

***Index Terms* — I2C Bus, FPGA, VHDL, Interface Model.**

I. Introduction

THE I^{2}C bus has become the de-facto world standard that is now used in different ICs. By employing the I^{2}C protocol in a design, much of the auxiliary support circuitry such as address decoders and standard logic gates needed for other communication methods can be eliminated. In an I^{2}C bus, there is no central server to resolve data conflicts. The collisions are prevented using the wired-and configurations of the serial data (SDA) line and the serial clock (SCL), and data loss is prevented by the fact that every byte on the SDA line has to followed by an acknowledge[1]. The network interface design presented in this paper can be used to interconnect any number of devices on a network efficiently as long as the total capacitance limit is not exceeded. The model can be used as a master or as a slave or both. In [2], the said work was implemented on Modelsim 6.0 IIIa. Here, we have shown the implementation on Quartus II 6.0.

Department of Electronics and Tele-Communication Engineering, Jadavpur University, Kolkata-700 032. INDIA.
Email: pvwn@ieee.org , mitamadhu123@gmail.com , arindam_3110@yahoo.co.in , snehasishetce@yahoo.co.in , robnon@ieee.org

The important I2C bus specifications [1] are described in Section II. The implementation methods are given in Section III and finally, simulation result and conclusion are given in Section IV.

II. I^{2}C Bus Specification

The I^{2}C bus is built around a two-wire serial bus, SDA (serial data) and SCL (serial clock). Each device is recognized by a unique address, and can operate either as a transmitter or a as a receiver. The I^{2}C master is the device that initiates a transfer and generates the clock for the same[3]. Any device addressed by the master is the slave. If more than one master attempts to transmit in unison, there will be a conflict. The I^{2}C specification solves this conflict by its arbitration process. Before explaining the arbitration process, the basic I^{2}C characteristics are given below.

A. I^{2}C Characteristics

The SDA and SCL lines are bi-directional lines connected to a positive voltage supply through a pull-up resistor. The bus is free when these lines are 'high'. The data on the SDA line is valid only when the SCL line is 'low'. During data transfer, the master generates the START and STOP conditions, which are unique conditions and are shown in Fig.1.

B. Data Transfer

Data bits are transferred after start condition, the transmission is byte oriented (8 bits + one acknowledge bit) where every data is placed on SDA line. After transferring each byte, the master frees the SDA line for the slave to send the acknowledge bit. The slave sends the ACK by pulling down the SDA line low. In the next clock pulse, master sense the SDA line. If the slave cannot accept the data, it sends NAK (No-Acknowledge) by leaving SDA line high. The master senses the NAK and can either stop the transfer or initiate a restart. The complete data transfer operation is shown in Fig.1.

C. Addressing Format

The multiple devices on the I2C bus can be differentiated using their addresses. The I2C device can be addressed using a 7-bit addressing or a 10-bit addressing. The 7-bit addressing format is described as we have used this format in our network interface model. The first byte sent on the I2C bus after the start is usually an address byte. The bits 7 through 1 of the address byte carry the I2C address Information and the last bit, bit 0 determine if the I2C operation will be a read or write. A zero in bit 0 tells the slave that the master will be writing data to the slave device, and conversely a 1 in the least significant bit tells the slave that master will read the information from the slave. Only the device that contains a match for the first seven bits will ultimately respond master.

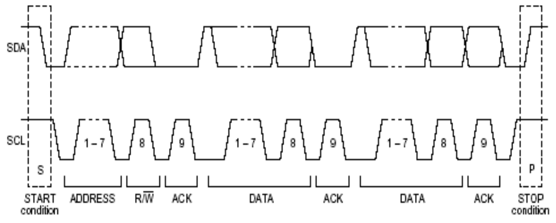


Figure 1. The complete Data Transfer operation

D. Arbitration

Arbitration takes place on the SDA line, while the SCL line is at the HIGH level, in such a way that the master transmits a HIGH level, while another master is transmitting a LOW level, will switch off its DATA output stage because the level on the bus doesn't correspond to its own level. Arbitration can continue for many bits. Its first stage is comparison of the address bits. If both the masters are trying to address the same device, arbitration continues with comparison of the data-bits if they are master-transmitter or acknowledge-bits if they are master-receiver. Because address and data information on the I2C-bus is determined by the winning master, no information is lost during the arbitration process. A master that loses the arbitration can generate clock pulses until the end of the byte in which it loses the arbitration. The winning master's address and data are the only valid items on the I2C bus and nothing is lost in the arbitration process.

III. Implementation Of The Proposed Interface

The network interface device we have designed consists of a Master section and a Slave section[2] as shown in Fig. 2. The internal signals have not been shown for the sake of simplicity. The device can function as a master or a slave depending on the stimuli provided. In this design, for ease of FPGA Coding, we have modified the method of generation of START and STOP conditions. The data initiation and termination signals are generated and conveyed to all the ICs in the network through a common BUS line. All the BUS lines of the devices have a wired-and configuration likes the SDA and SCL lines. The I^2C bus is free if the BUS is at a 'high' state. Any master attempting to transmit first senses whether the BUS is at a 'high' state. If the BUS is 'high', the master immediately captures the I^2C bus by pulling down the BUS line to a 'low' state. If two masters attempt to capture the bus at the same time, then the winning master is decided by following the I^2C arbitration logic.

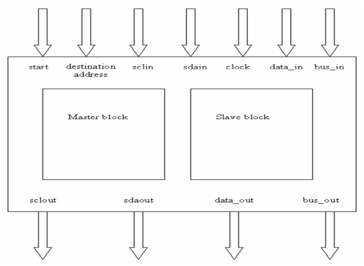


Figure 2. Functional block diagram of the Network Interface

The interfacing device contains two main blocks – the master and the slave. The details of the master and slave blocks and their functioning are described in Section A and B respectively. When the device which is being addressed is determined after the transmission of the first byte (address byte) is over, it starts to act as a slave till the master terminates the data transmission. If two masters try to transmit data simultaneously, and it so happens that the winning master is actually addressing the losing master, the losing master can immediately switch on to the slave mode without any loss of data of the winning master.

A. Master Block

The Master has five main functional blocks: a) Initiator b) Address Block c) Write Block d**)** Read Block e) Clock generator**.** The functioning of the master block is given below:

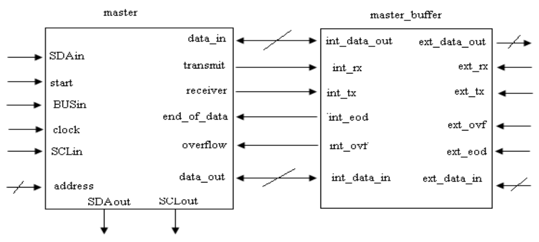


Figure 3. Block diagram of the Master

Step 1. When the master has to perform any data transfer, the Initiator senses whether the bus is free. If the bus is free, it pulls the BUS line to a 'low' and enables the clock generator which generates the clock for data transfer, and the Address Block.

Step 2. The Address Block sends each bit of the address byte on the SDA line on falling edges of the SCL and also checks whether the data on the SDA line is what it has sent. If the data on the SDA line does not match with the address bits, the Address Block senses that it has lost control of the bus to another master and sends the signal control_add to the Initiator which then frees its SDA, SCL and BUS lines. After completion of address sending, the Address Block frees the SDA line for the slave to send acknowledge (ACK). If the slave sends an ACK, then it sends a signal over_add to the Initiator which then enables the read or write block. If the slave sends a NAK, it sends a signal error_add to the Initiator which then terminates the transfer.

Step 3. The Write Block transfers data to the slave on the SDA line at falling edges of the SCL. It also checks whether the data on the SDA line is the same as the data it has sent, and waits for the ACK from the slave after transfer of each byte. If there is a NAK from the slave, it reports to the Initiator which then terminates the transfer.

Step 4. The Read Block reads data from the slave at falling edges of the SCL. After reception of each byte, it sends an ACK to the slave or a NAK if it wishes to terminate the transfer.

B. Slave Block

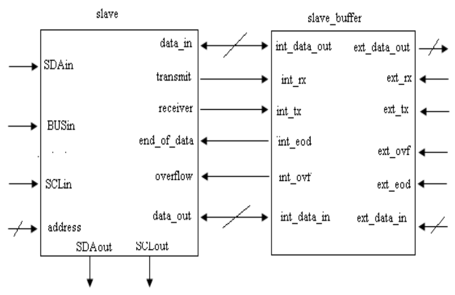


Figure 4. Block diagram of the Slave

The Slave Block has four main functional parts: a) Monitor b) Address Block c) Receiver d) Transmitter. The functioning of the slave block is given below:

Step 1: The Monitor continuously senses the state of the BUS line. If it senses that the bus line has been pulled to a ‘low’ state, it senses that a master has captured the bus. It then enables the Address Block.

Step 2: The Address Block checks whether the address sent by the master on the SDA line matches with it address. In case of a match, it sends the add_match signal to the Monitor which then enables the transmitter or receiver. In case of no match, the Monitor waits for the next initiate-transfer condition.

Step 3: The Receiver is responsible for receiving the data from the master and sending the ACK (or NAK) to the master. It reads the data at positive edges of the SCL. On detecting a release of the bus by the master, the receiver is disabled by the monitor.

Step 4: The Transmitter sends data to the master on the SDA line at positive edges of the SCL. It also checks for the ACK (or NAK) sent by the master. It terminates the data transfer on receiving a NAK from the master.

On the basis of the proposed interface, Algorithmic State Machine for the Slave is designed and is shown in Fig.5. Due to space limitation, the state machine for the Master is not shown here. Further, we have developed the Register Transfer Language for both Master and Slave, but not shown here due to space limitation.

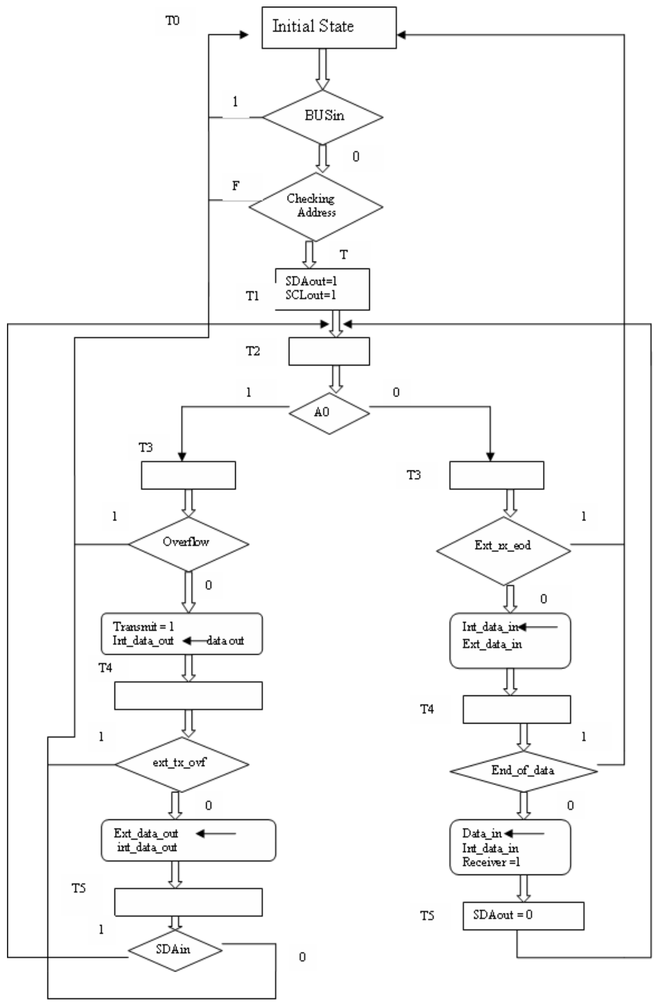
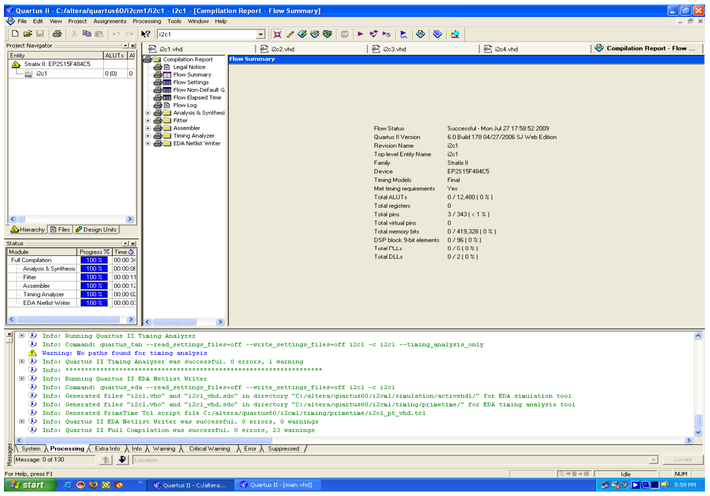


Figure 5. Algorithmic State Machine of the Slave

IV. Simulation Results And Conclusion

We have used Quartus II 6.0 to create a VHDL[4] model of the I^2C device. We have simulated a network by connecting three different instantiations of the I^2C model. Here EP2S15F484C5 device in the family of Stratix II is used to stimulate the network. The flow summary of the device1 is shown in the Fig 6. Here total number of pins used is three; SDA, SCL and BUS are bidirectional pins. The data in the buffer of the slave device 3 is the same data sent by the master device1. Thus our interface model is in accordance with the I^2C specifications. Since, we have created the model using FPGA; it will allow rapid prototyping of the interface model for large scale communication designs involving ICs. Some architecture of that particular microcontroller, for example, to implement I^2C on the PIC microcontroller [5,6] one has to know about the structure of the registers SSPCON, SSPSTAT and SSPBUF. In our interface model, setting up the network and having the devices talking to each other requires minimal coding. Information can be exchanged over the network just by issuing certain signals. Also, an FPGA based I^2C interface model is much faster than the microcontroller interfaces.



Flow Status	Successful - Mon Jul 27 17:58:52 2009
Quartus II Version	6.0 Build 178 04/27/2006 SJ Web Edition
Revision Name	i2c1
Top-level Entity Name	i2c1
Family	Stratix II
Device	EP2S15F484C5
Timing Models	Final
Met timing requirements	Yes
Total ALUTs	0 / 12,480 (0 %)
Total registers	0
Total pins	3 / 343 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

Module	Progress %	Time
Full Compilation	100 %	00:00:34
Analysis & Synthesis	100 %	00:00:06
Fitter	100 %	00:00:11
Assembler	100 %	00:00:12
Timing Analyzer	100 %	00:00:02
EDA Netlist Writer	100 %	00:00:03

```
Info: Running Quartus II Timing Analyzer
Info: Command: quartus_tan --read_settings_files=off --write_settings_files=off i2c1 -c i2c1 --timing_analysis_only
Warning: No paths found for timing analysis
Info: Quartus II Timing Analyzer was successful. 0 errors, 1 warning
Info: *******************************************************************
Info: Running Quartus II EDA Netlist Writer
Info: Command: quartus_eda --read_settings_files=off --write_settings_files=off i2c1 -c i2c1
Info: Generated files "i2c1.vho" and "i2c1_vhd.sdo" in directory "C:/altera/quartus60/i2cm1/simulation/activehdl/" for EDA simulation tool
Info: Generated files "i2c1.vho" and "i2c1_vhd.sdo" in directory "C:/altera/quartus60/i2cm1/timing/primetime/" for EDA timing analysis tool
Info: Generated PrimeTime Tcl script file C:/altera/quartus60/i2cm1/timing/primetime/i2c1_pt_vhd.tcl
Info: Quartus II EDA Netlist Writer was successful. 0 errors, 0 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 23 warnings
```

Figure 6. Flow Summary of the Device 1

ACKNOWLEDGEMENT

This project is supported by University Grants Commission (UGC), Ministry of Human Resource Development(MHRD) under its Major Research Project(MRP) scheme, entitled "FPGA Based Interface Model for Scale-free Computer Network", F.No. 33-92/2007(SR) dated 29th February, 2008

REFERENCES

[1] THE I^2C BUS SPECIFICATION VERSION 2.1 January, 2000, Philips Semiconductors.

[2] P.Venkateswaran, A. Saynal, S. Das, S.K Saynal and R. Nandi , 'FPGA Based Efficient Interface Model for Scale-Free Computer Networking using I2C Protocol' , 15th.intl'conf on computing – CIC 2006, Proc. Research in Computing Science: Special Issues – Advances in Computer Science & Eng., ISSN 1870 – 406, pub .National Polytechnic Institute, Mexico, Vol. 23 , pp 191 -198 , Nov. 21-24,2006.

[3] Fred Eady, Networking and Internetworking with Microcontrollers, Elsevier ,2004.

[4] J. Bhasker, A VHDL Synthesis Primer, BS Publications 2nd Edition, 2003, pp. 132.

[5] TMS470R1x Inter-Integrated Circuit(I^2C) Reference Guide, Texas Instruments (SPNU223).

[6] Design Ware Inter-IC (I^2C) VIP Data book, Version 1.10a, February 18, 2005, Synopsis.