FPGA Based Efficient Interface Model for Scale-Free Computer Network using I2C Protocol

P.Venkateswaran, Arindam Sanyal, Snehasish Das, S.K.Sanyal and R.Nandi

Electronics & Tele-Communication Department, Jadavpur University, kolkata-700 032 INDIA
pvwn@yahoo.co.in, arindam_3110@yahoo.co.in, snehasishetce@yahoo.co.in,
s_sanyal@ieee.org, robnon@ieee.org

Abstract. Devices communicating with each other over a serial bus must have some protocol to avoid data loss, as well as enabling faster devices to communicate with slower ones. In a network, there may be several electronic modules across which a communication link has to be established. There must be some algorithm to solve conflicts between the contending nodes in the network. In this design, we have implemented an efficient network interface model which can prevent data loss due to collisions. The proposed interface model follows the I2C protocol with a slight change in the manner in which the data transfer is initiated or terminated. The I2C bus is a true multimaster bus which includes arbitration safeguards against data collisions. The interface model described can be used to connect any number of devices to a network.

1 Introduction

The I^2C bus has become the de-facto world standard that is now used in different Integrated Circuits(ICs). The advantages of using I^2C are numerous and by employing the I^2C protocol in a design, much of the auxiliary support circuitry such as address decoders and standard logic gates needed for communications can be eliminated. In an I^2C bus there is no central server to resolve the data conflicts. The collisions are prevented using the wired-and configurations of the Serial DAta (SDA) line and the Serial CLock (SCL) line, and the data loss is prevented by the fact that every byte on the SDA line has to be followed by an acknowledge. The important I^2C bus specifications[1] are described in Section 2. 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 a master or a slave or a combination of both. The model can be used as an initiator to start/stop all possible data transfers. The model can be used as target (slave) device which detects start/stop conditions and perform data transfer according to the master's request. The model is capable of handling traffic and detecting faults, if any, on the bus. Moreover, if data transfer is to be done intermittently, the chip can be used in parallel for different applications. The implementation[2] methods are given in Section 3 and the performance results of the proposed model using simulation is given in Section 4.

Advances in Computer Science and Engineering.
Research in Computing Science 23, 2006, pp. 191-198

Received 07/07/06
Accepted 03/10/06
Final version 12/10/06

2 I2C Bus Specification

The I^2C 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 as a receiver. The I^2C master is the device that initiates a transfer and generates the clock for the same. Any device addressed by the master is the slave. If more than one master attempts to transmit at the same time, there will be a conflict. The I^2C specification resolves this conflict by its arbitration process. Before explaining the arbitration process, the basic I^2C characteristics is given in Section 2.1.

2.1 I2C 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. A high to low transition on the SDA line while the SCL line is high, indicate the START condition, while a low to high transition on the SDA line while the SCL line is high, indicate the STOP condition (Fig. 1). The START and STOP conditions are always generated by the master. The bus is considered to be busy if a START condition is generated.

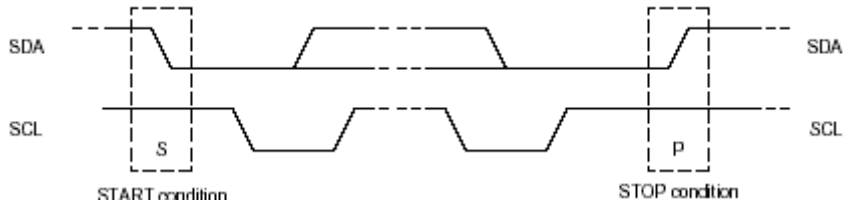

Fig. 1. START and STOP condition

2.2 Acknowledge

Every data put on the SDA line is 8-bits long. Each byte has to be followed by an acknowledge bit (Fig.2).

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 the SDA line low. The master reads the SDA line in the next clock pulse and senses the acknowledge. If the slave cannot accept the data due to some reason, it sends a NAK (No-AcKnowledge) by leaving the SDA line high. The master sends the NAK and can either stop the transfer or initiate a restart

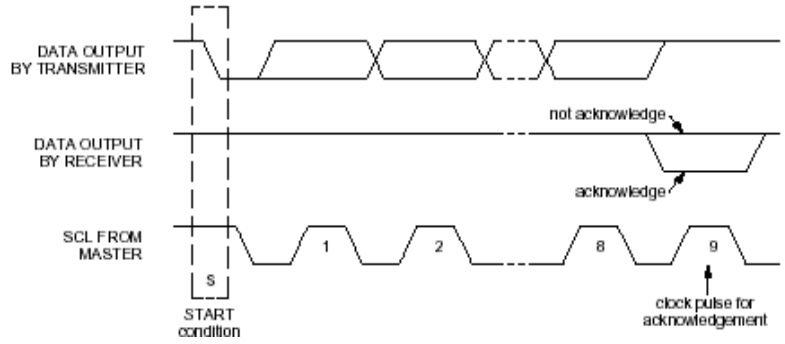

Fig. 2. Acknowledge on the I^2C bus

2.3 Clock Synchronization

The SCL lines of the I^2C bus have wired-and connection. This plays an important role in clock synchronization. The wired-and connection implies that the I^2C device with the lowest low period will hold the SCL line low. If the slave is busy to service an internal request and needs more time to respond to the master's request, it can pull the SCL low. As long as the SCL is held low by the slave, the master cannot bring it to the high state. This is known as 'clock stretching'.

2.4 Arbitration

The SDA lines also have wired-and configurations like the SCL lines. This fact is used in arbitration of the I^2C bus. A master may transfer data only if the bus is free. Now two or more masters may attempt to transfer data at the same time by initiating a START condition simultaneously. Arbitration takes place on the SDA line while the SCL line is at the high level such that the master transmitting a high while another master is transmitting a low, will lose the control of the bus. Arbitration may continue for several bits. Its first stage is the comparison of the address bits. If both the masters are trying to address the same slave, then the arbitration will continue into the data bits till one of the masters attempt to transmit a high while another is transmitting a low. The losing master will then release its SDA line and will revert to the slave mode if it is being addressed by the winning master. The winning master's address and data are the only valid items on the I^2C bus and nothing is lost in the arbitration process.

2.5 Addressing Format

The multiple devices on the I^2C bus can be differentiated using their addresses. The I^2C devices 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 inter-

face model. The first byte sent on the I^2C bus after the start is usually an address byte. One exception involves sending a "general call" address following the start condition. The "general call" addresses everyone on the I^2C bus. Any device not wishing to listen to the "general call" can do so by not sending an acknowledge to the master. The addressing scheme is shown in Fig. 3.

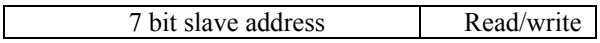

Fig. 3. The first byte after START

The bits 7 through 1 of the address byte carry the I^2C address information and the last bit, bit 0 determines if the I^2C 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 lsb (least significant bit) tells the slave that the master will read information from the slave. Only the device that contains a match for the first seven bits will ultimately respond to the master

3 Implementation of the Proposed Interface

The proposed network interface consists of a master section and a slave section(Fig. 4). The internal signals have not been shown for the sake of simplicity. The details of the master and slave blocks are shown in figs. 5 and 6 respectively. The device can function as a master or a slave depending on the stimuli provided. In this design, we have modified the I^2C protocol by changing the generation of START and STOP conditions. In the I^2C protocol the START condition is denoted by the master pulling down the SDA line while the SCL is high. Similarly, a master signals end of transmission by taking back the SDA to high state while the SCL is high. We have made the interface respond only to the rising and falling edges of the SCL[3]. Checking for the START or STOP conditions involves continuously monitoring the SDA line for changes while the SCL is high. This implies that the interface will be busy for half of the clock period doing practically nothing. So we have used an alternate method to utilize the high state of the clock in performing data shifting operations between the user and the interfacing device so that data can be transferred between the communicating ICs with minimum of delay. The data initiation and termination signals are instead generated and conveyed to all the ICs in the network through a common BUS line. The interface is a three wire device consisting of the SDA, SCL and the BUS lines. The state of the bus at any time is reflected by the BUS line. If a transfer is in progress, the BUS line is at a low state to signify that the bus is busy. The master senses the state of the I^2C bus through a signal called BUS. All the BUS lines of the devices have a wired-and configuration like 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. The START and STOP signal generation and detection can be performed in another way which follows the I^2C specification. This involves gen-

eration of another clock signal which is delayed by a fixed time period from the SCL signal. This delayed clock signal is locally generated by all the devices connected to the network from the SCL signal which is clocked by the master. Then instead of having to continuously monitor the SDA line for changes when the SCL is high, the SDA has to be checked for a change only after a fixed time from the rising edge of the SCL. But this method involves generation of very accurate delays and it also has to take the propagation delay of the SDA and SCL signals into consideration. From our experiments we have seen that delay in SDA is usually not equal to the delay in the SCL line and hence this second method requires greater coding effort. The unequal delays are also to be expected as the SDA changes only a certain time after rising or falling edges of the SCL. The master and slave are further divided into smaller functional units. The details of the master and slave blocks and their functioning are described in Section 3.1

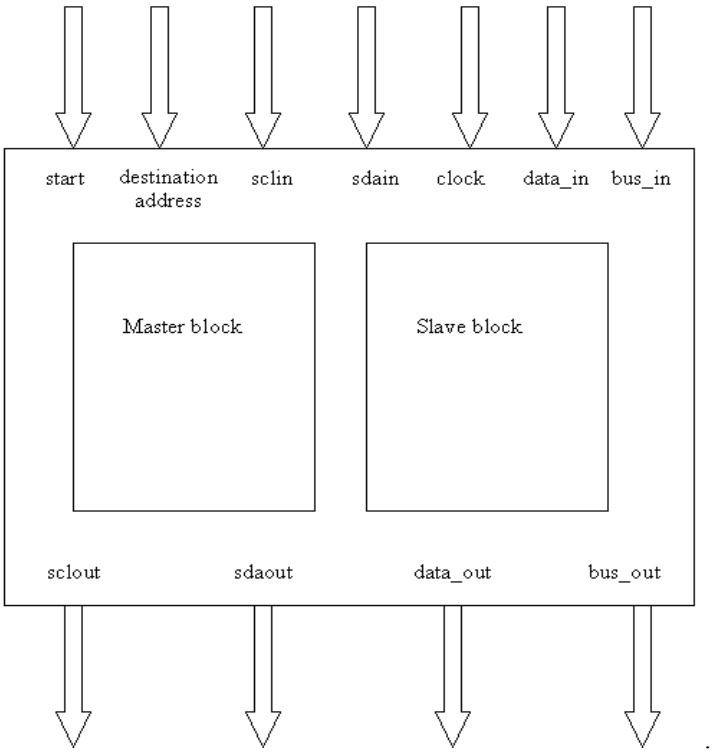

Fig. 4. Functional block diagram of the network interface

3.1 Master Block

The block diagram (Fig. 5) below shows the details of the master block.

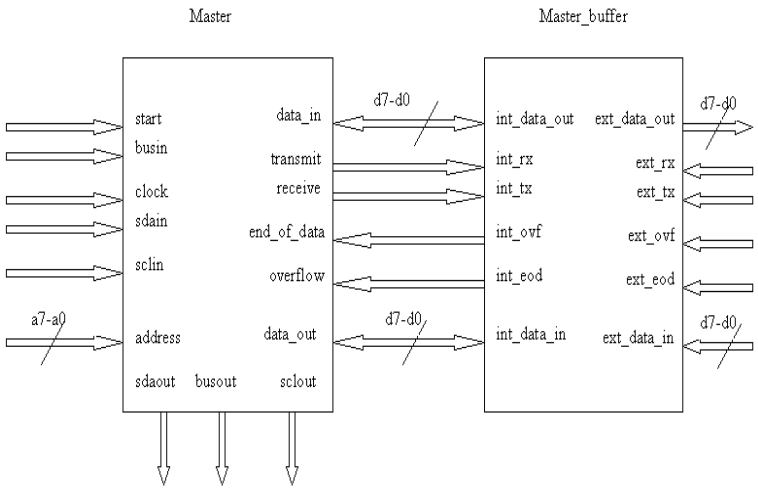

Fig. 5. Functional block diagram of the master

The master block has its own buffer where data to transmitted can be stored by the user(external data) or the data received from the slave can be stored by the master (internal data). The buffer has a fixed capacity, and if the master or the external user tries to send more data to the buffer while it is full, the buffer will generate overflow signal(ext_ovf or int_ovf). Similarly, the buffer will generate the end_of_data signal(ext_eod or int_eod) if any attempt is made to retrieve data from the buffer while it is empty. The master has five main functional units: a) Initiator b) Address Block c) Write Block d**)** Read Block e) Clock generator**.** The functioning of the master block is given below:

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 NAK if it wishes to terminate the transfer.

3.2 Slave Block

The block diagram (Fig. 6) below shows the details of the slave block

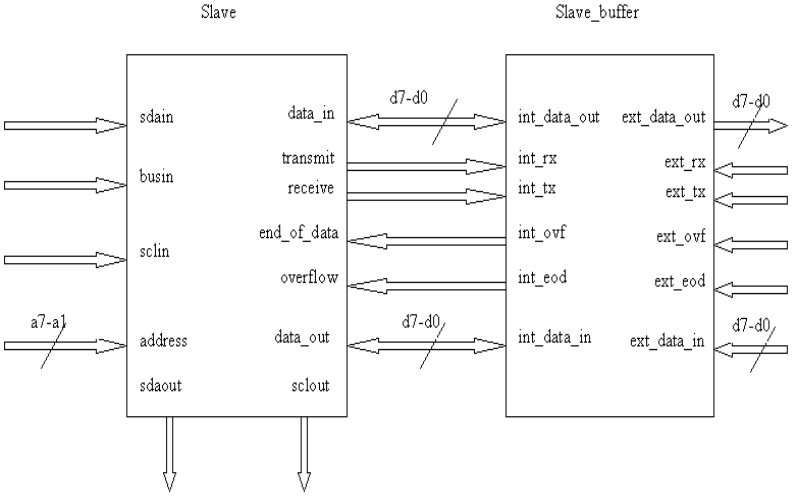

Fig. 6. Functional block diagram of the master

The Slave Block has four main functional units: (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.

4 Simulation Results and Conclusion

We have used Modelsim 6.0 IIIa to simulate the I^2C devices. We have simulated a network containing three devices. Each of the devices have their own unique address by which they can be addressed. The address of the devices are given below:

Table 1. Address map of the devices

Device name	Device address
dev1	1111100
dev2	1111101
dev3	1111110

We have simulated the traffic in the network by using a testbench[4]. The masters of devices 1 and 2 were given a 8-bit data and their destination addresses were set to the address of device 3. Both the masters were given the start signal at the same time so that a conflict occurs. Due to the proposed I^2C arbitration specifications, one of the masters (device 1) wins control of the bus, and its data is transferred to the slave of device 3 successfully. This can be seen from the simulation results given below (Fig. 7).

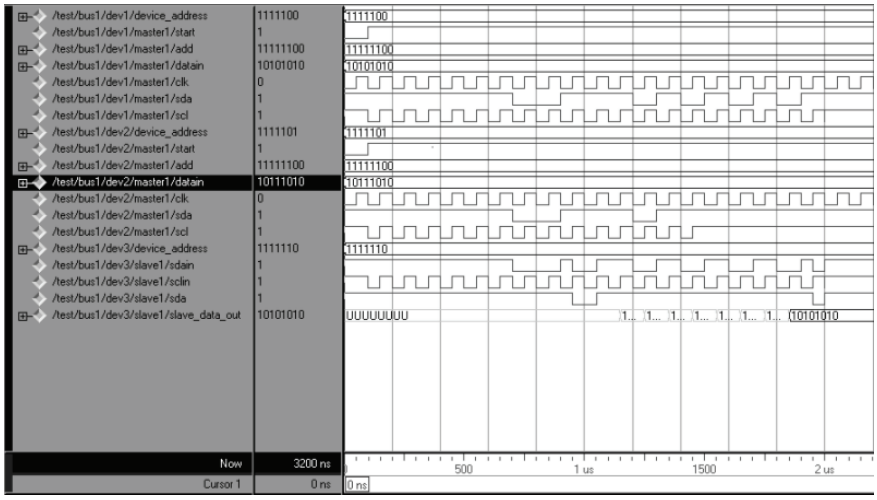

Fig. 7. Simulation waveform

References

1. THE I^2C BUS SPECIFICATION VERSION 2.1 January 2000, Philips Semiconductors
2. *IEEE Standard VHDL Language Reference Manual*, IEEE std 1076-1993 (revision of 1076-1987)
3. Fred Eady, *Networking and Internetworking with Microcontrollers*, Elsevier ,2004
4. J Bhasker *A VHDL Synthesis Primer,* BS Publications 2nd Edition, 2003 , pp. 132