International Journal of Applied Engineering Research.
ISSN 0973-4562 Volume 10, Number 10 (2015), pp. 26245-26257

http://www.ripublication.com

Implementation of various interfacing techniques between an FPGA and a microcontroller targeting complex cost effective embedded applications

Anusha Janardhan and Jayanta Biswas
School of Computing, SASTRA University,
E-mail: anushaj144@gmail.com, jayantabiswas@ict.sastra.edu

Abstract

Embedded applications are complex and diverse in nature. Design of complex embedded systems depends on the requirement of the specific application. Embedded systems design involves microcontrollers, processors and FPGA/ CPLD as part of design components. Requirements of the embedded system application determine the role of microcontroller/ FPGA as *master* or *slave*. Communication strategies between FPGA and microcontroller also need to be implemented as a part of the design. In this proposal, various cost effective communication strategies between microcontroller and FPGA are designed, implemented and tested. Microcontroller executes the role of *master* or *slave* and FPGA executes the opposite role corresponding to the microcontroller. The objective of the proposal is to implement and test the communication frame work between FPGA and microcontroller. Interfacing techniques discussed in this proposal find applications in embedded voice and data transmissions in high end systems and also in large scale chassis based embedded voice/video platforms. All communication strategies have been implemented and verified on entry point Altera Cyclone II family of FPGA devices, hence giving way to a cost effective solution.

AMS subject classification:
Keywords:

1. Introduction

Chassis based embedded platforms or any high end embedded application consists of multiple computing components. Each component is associated with different applications with different application interfaces. Hence establishment of a high speed and efficient communication between these applications becomes crucial.

Various protocols have been used for communication amongst multiple design components in embedded chassis based applications as in products referred from [6], [7] and [8]. These consist of external synchronizers that add to area and cost overhead. An FPGA interface device that includes an on-board FPGA, a microcontroller, and a memory having first and second pages has been described in [1]. A multi-purpose interface between a host computer and an FPGA, whose data lines are used to communicate unidirectional configuration data into the FPGA is described in [2]. The data is clocked by the host computer using the data strobe signal line to clock data into the FPGA. An interface between a real microcontroller and a virtual microcontroller (an FPGA emulating a microcontroller) is tested in [3] and [4]. A user-programmable integrated circuit that includes over the same silicon die a set of programmable logic cells, microprocessor which is used both for controlling and configuring the whole system and an optimized interface between these blocks are described in [5].

In this proposal, a number of strategies have been brought together and compared based on the application requirements. Thus a correct choice of interfacing strategy may be made on the basis of this comparison. Moreover, all the strategies discussed have also been optimized with respective to cost, speed and area. This forms the novelty of the proposal as these issues have not been addressed together before. In this proposal, three such strategies have been developed and tested namely,

- SPI (Serial Peripheral Interface)
- TDMA channel (T1/E1) serial interface
- Interface between FPGA and a bare microcontroller using a state machine

One of the components is be modelled as *master* (server) and the other as *slave* (client). Different server and client applications may interact over multiple FPGAs and microcontrollers with above mentioned techniques. This aids in providing very high availability of the system. Interface between components with different clock is ensured by use of dual port RAMs between components.

Each of the strategies suffices different requirements in complex embedded applications that contain multiple FPGAs and microcontrollers and their corresponding design components. Choice of a particular strategy depends upon level of integration of the target system.

High end systems with very high levels of integration may require SPI for tasks like programming, debugging or loading. A Serial Peripheral Interface between an FPGA and a microcontroller also finds applications in interrupt driven systems. Scalable voice telephonic applications require SPI for interactions with flash memory, SD cards and other components that need serial support for data transfers. Data and control buses that run across complex chassis based systems require SPI to communicate with different computing elements present on the chassis.

A TDMA (Time Division Multiple Access) channel interface allows multiple users to access the same frequency channel simultaneously. A channel is divided into multiple time slots, where each slot carries data for a different user. Each slot is of one byte and thus carries one byte information to different users. In this proposal, a T1 (T-carrier) standard is used on front end to communicate with multiple E1 (E-carriers) present on back plane. T1 is an American carrier standard with a frequency of 1.544 Mbit/s with 24 user channels while E1 is a European carrier standard with a frequency of 2.048 Mbit/s with 32 user channels. In this proposal, T1 and E1 carriers are used to communicate between different users on front end and back plane. *Sync* signal from front end to back plane remains at same frequency. Different channels on T1 side are configured for transmission and reception of data from different E1 users. Such an interface is extremely useful in systems that employ triple play services such as, voice, video and data.

The third strategy is designed for the case when the target system is very superficial. The client consists of a bare microcontroller which is not fully developed and is devoid of any interface with other components. In such cases, a high speed interface may be developed just by designing an FSM (Finite State Machine) which may be used to communicate with other components in the system.

2. Interfacing techniques

2.1. SPI (Serial Peripheral Interface)

SPI is developed between a server and a client. Server acts as the *master* and client acts a *slave*. The server is associated with an application through which the user may transmit and receive data from the client. SPI forms a synchronous serial communication interface that consists of four primary signals namely, *SCK* (Serial Clock), *CS* (Chip Select), *MISO* (*Master* Input *Slave* Output) and *MOSI* (*Master* Output *Slave* Input). *CS*, *SCK* and *MOSI* are output signals from server (*master*) while *MISO* is an 8-bit bus input to server from client. Apart from these signals, a *master* clock CLK is provided by the server.

Fig 1 shows the basic block diagram of an SPI. The server-application interface consists of *RD* (read), *WR* (write), *ACK* (acknowledge) and *RST* (reset) signals. Apart from these signals, the interface also consists of two buses, *DATA-IN* that is an 8- bit input bus to the server and *DATA-OUT* that is an 8-bit output bus from the server to the server application.

CS is made low for certain duration during which transmit of data from the *master* takes place. Eight pulses of serial clock (*SCK*) are generated in the duration when *CS* is made low as shown in Fig 9. Data to be transmitted from the server is sampled on positive edge of *SCK* and data to be received at the server side is sampled on negative edge of *SCK*.

During server transmit operation, the data to be transmitted is sent from the application to the server via *DATA-IN* bus. Initially, *FE* byte is placed on *MOSI* indicating the start of the operation. *FE* byte is followed by actual data bytes that need to be transmitted that are available on *DATA-IN* bus. On client side, *MOSI* is sampled on the negative edge of *SCK* and in this way the serial data available on *MOSI* is converted into data bytes on the *slave* side.

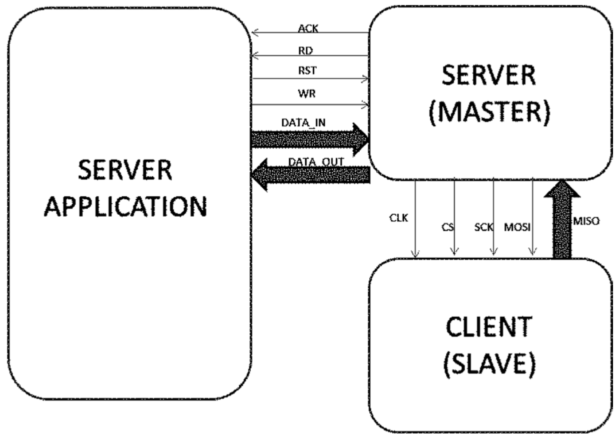

Figure 1: SPI block diagram showing interface between server and client modules.

During the server receive operation, data is sent from the client side through the 8-bit bus *MISO*. This data is converted into serial data and then received at the server application side.

2.2. TDMA channel (T1/E1) serial interface

A T1/E1 serial channel interface allows multiple services to be carried out at different time slots. Out of the 24 channels present on the front end (T1 side), each channel can be assigned to a separate user at the back end that consists of multiple E1 users. Hence one channel or a set of channels on the T1 side is mapped to a unique E1 on the back plane. Thus the 0th channel on T1 side may be mapped to 1st E1 channel, 1st channel on the T1 side may be mapped to 2nd E1 channel, and so on. Thus in this way, data may be transmitted and received across 24 E1 users in the back plane in one second. Hence this interfacing technique proves out to be highly efficient.

The clock frequencies on T1 and E1 sides are different and hence synchronization of data and avoiding its overflow becomes the most crucial and difficult part in the design of this interface. Hence a dual port RAM is used in this case, to avoid loss of data and at the same time ensure a high speed interface. This aids us in building a highly cost effective strategy as no external synchronizers are used for this purpose. Dual port RAMs are readily available in Altera Cyclone II family of FPGAs.

Fig 2 shows the basic block diagram of T1/E1 channel serial interface consisting of a single user on E1 side. The same may be extended for multiple user applications on E1 side. Server-application interface consists of *WR* (write), *RD* (read), *RST* (reset), *TXD* signals. The server application sends the data to be transmitted to E1 side serially, by placing it on the appropriate channel on the TXD line. Similarly, the client-application interface consists of *WR*, *RD*, *RST* and *RXD* signals. Client module provides *RXD* signal as output to the client application. *RXD* signal consists of the serial data that has been placed on the corresponding E1 channel after transmitting from the T1 side. A dual port RAM is needed because the server and the client operate at different clock

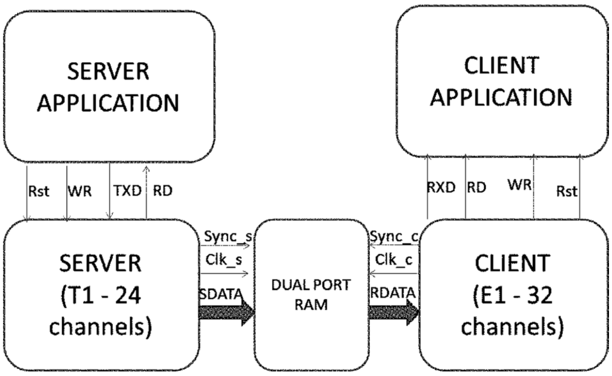

Figure 2: T1/E1 interface block diagram.

frequencies as mentioned earlier. Server provides *CLK-S* (server clock) and *SYNC-S* (server synchronization signal) to the dual port RAM. Similarly, client provides *CLK-C* (client clock) and *SYNC-C* (client synchronization signal) to the dual port RAM. Serial data on the TXD line is placed as parallel data bytes on the *SDATA* and this operation is synchronized with server clock or T1 clock. *SDATA* is an 8-bit bus input to dual port RAM. This data is read from dual port RAM into the *RDATA* and this operation is synchronized with client clock or E1 clock. Thus the completion of the whole operation depends upon the slower clock amongst the two clocks.

2.3. Interface between FPGA and a bare microcontroller using a finite state machine

In this interfacing technique, communication is established between an FPGA and a bare microcontroller based on the exchange of certain hand shaking signals followed by the transmission/reception of actual set of data. A bare microcontroller has input and output ports but has no prior interface to the outside world. Thus novelty of this model lies in the fact that an interface may be established even with a basic processor that is not fully developed and integrated. Moreover, this technique proves out to be highly efficient and cost effective as it may be developed with the use of any entry point FPGA and within a very short development time. Fig 3 shows the basic block diagram for interface using an FSM.

One FSM is designed each for the server and the client. FSMs are designed based on the flowcharts given in Fig 4 and 5. Fig 4 stands for the server-transmit/client-receive flowchart and Fig 5 represents the flowchart for server-receive/client-transmit mode. Initially certain fixed data bytes are exchanged and acknowledged on both sides as depicted by the flowchart. This is followed by transmission/reception of actual data bytes that need to be transmitted/received. The FSMs are designed using Verilog HDL on the basis of the flowcharts. System generated state diagrams for server and client are shown in Fig 6 and Fig 7, respectively. Thus a high speed and efficient interface may be developed by design and implementation of two simple state machines.

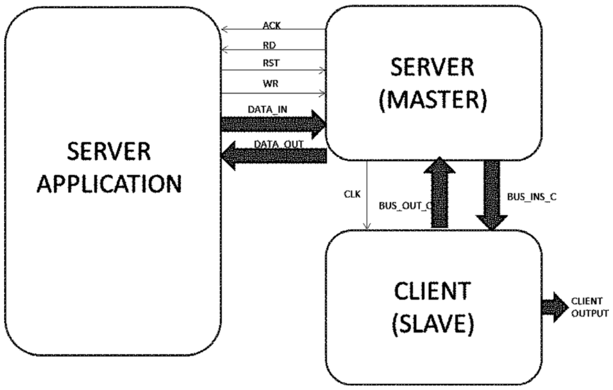

Figure 3: Block diagram for interface using an FSM.

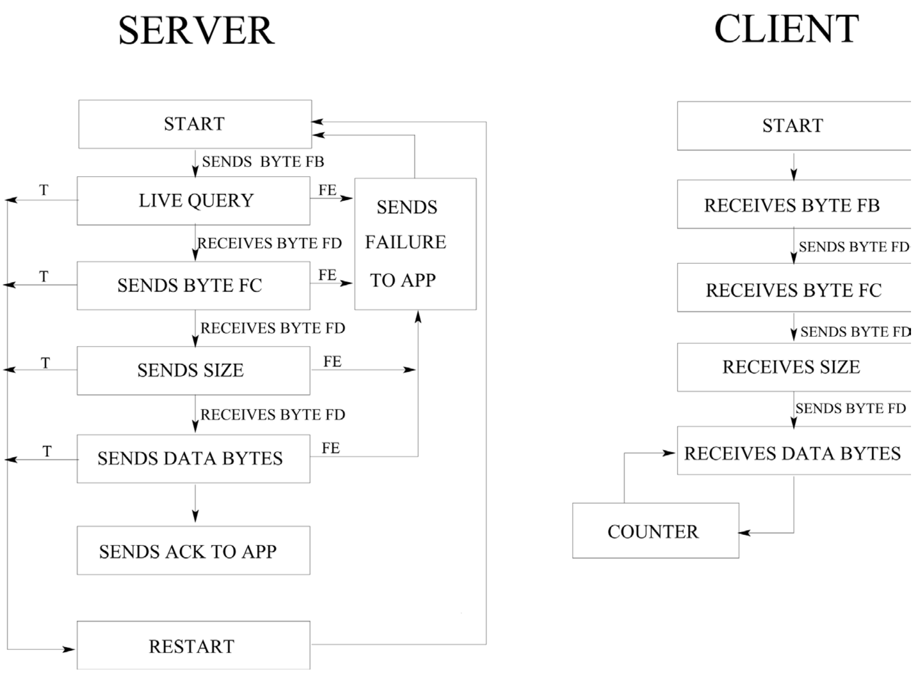

Figure 4: Server-transmit/Client-receive mode flow chart.

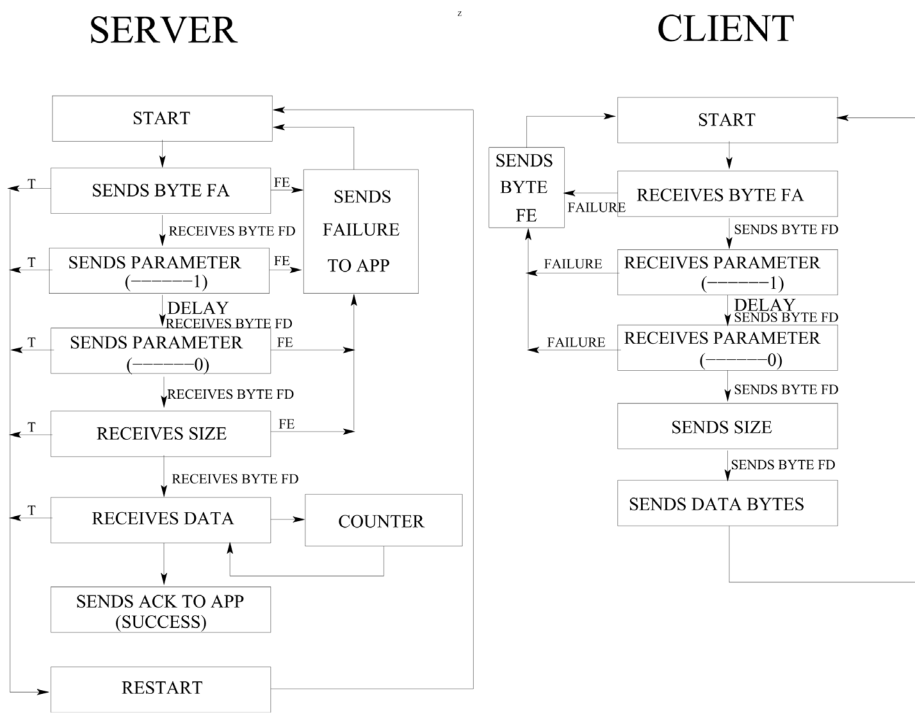

Figure 5: Server-receive/Client-transmit mode flow chart.

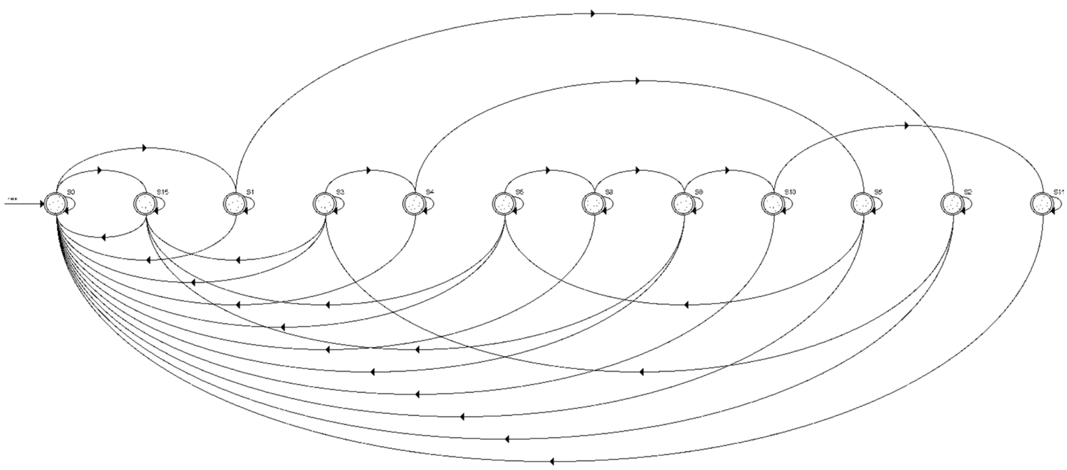

Figure 6: Master state diagram.

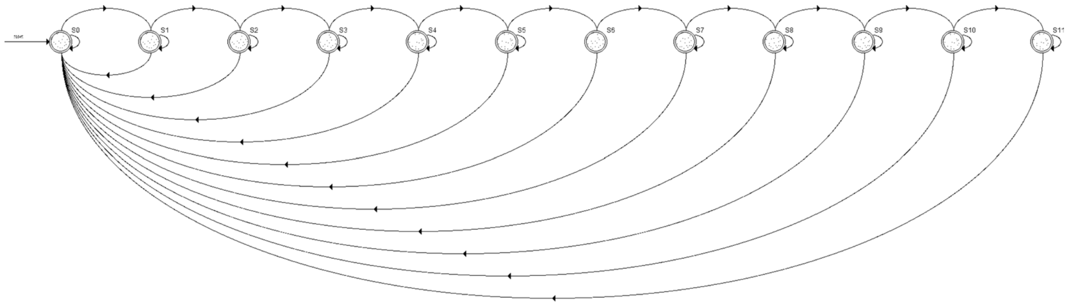

Figure 7: Slave state diagram.

3. Experimental Setup

Experimental setup for an interface consists of an Altera Quartus DE II board that consists of Altera Cyclone II FPGA, PIC 18F4525 microcontroller, a PIC add-on card, power supply and an oscilloscope. The FPGA acts as the server and the PIC acts as the client. The power supply is used to provide 5V *Vcc* to PIC add-on board. Oscilloscope is used to observe the output signals from FPGA or PIC. GPIO (General Purpose Input Output) ports present on the Altera DE II board are utilized for connections between FPGA board and PIC add-on card. The server module is designed by using Verilog HDL. ALTERA Quartus II v 12.1 is used for HDL entry. Block Diagram Format (BDF) file is used for establishing connections between the different modules as shown in Fig 8. The BDF file acts as the top level entity or sheet that connects all the modules into a top layer. The BDF file is designed and the project is compiled. The functionality of the different modules is initially verified by use of functional simulation. Functional simulation is done using Quartus Simulator.

Static timing analysis is done to verify that the modules meet the timing requirements when different constraints are specified. Altera Time Quest Timing Analyzer is used for this purpose. An SDC (Synopsys Design Constraint) file is written into the project. The SDC file is used to specify the timing constraints for the design. Various clock frequencies and input and output delays of the various paths are specified in the SDC file. The project is compiled again and Time Quest Timing Analyzer is used to verify if the setup and hold requirements of the design are met. This is implied by positive values for setup and hold slacks. The maximum frequency requirement is verified by the FMAX Summary. It is also checked that all paths present in the design are constrained. Timing simulation is carried out using Quartus Simulator to verify whether the design behaves in expected manner.

Thus functional simulation, static timing analysis and timing simulation for all modules are completed, followed by pin assignments. Pin assignments are done in such a way that the server application inputs present in the design are assigned to switches and the server application outputs are assigned to LEDs present in the first Altera DE II board. The same procedure is followed for configuring the client application inputs and outputs in the second Altera DE II board. The interface between the server and the client is

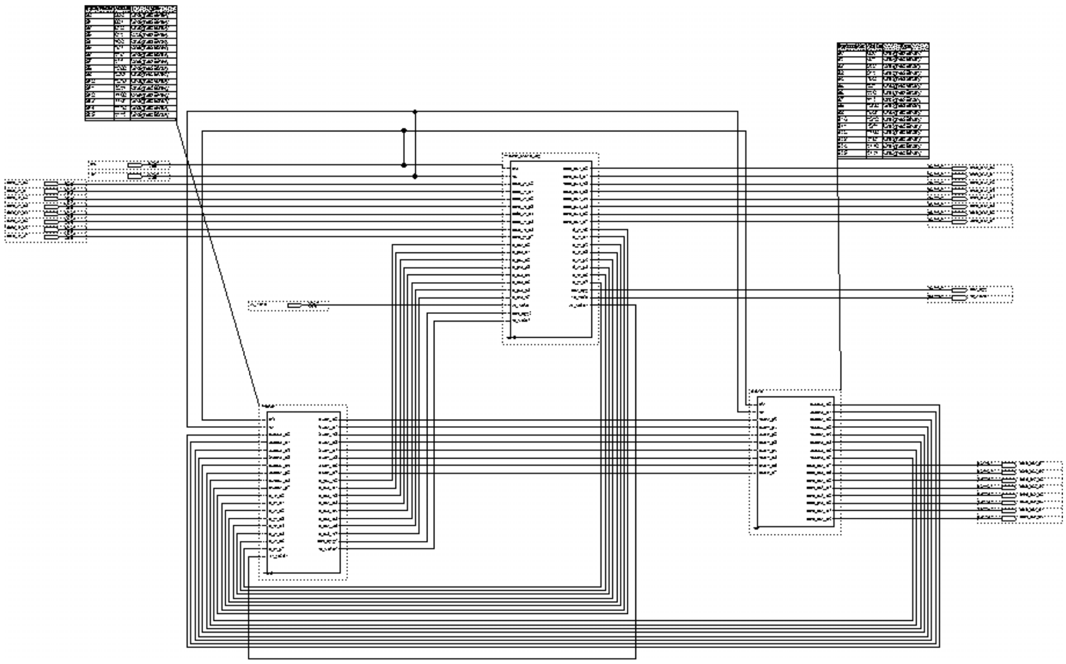

Figure 8: Block diagram schematic file (BDF) for interface using FSM.

Table 1: Resourc Usage Summary for SPI.

Logic Elements	
LUTs	23
Logic Registers	16

assigned to the GPIO ports present in the two boards. The GPIO ports of the two boards are connected via wires. The server application inputs are provided through switches present on DE II board while the outputs are observed through the LEDS present on DE II board. Hence, *rd*, *wr*, *ack* and the *data-out* signals are assigned to green and red LEDs. *Rst* and the *data-in* signals are assigned to push button switches.

Client setup consists of PIC18F4245 and PIC add-on card. The add-on board consists of voltage level translators and connectors in addition to a slot for placing the PIC microcontroller. Power supply is connected to the *Vcc* and *GND* pins of add-on board. The power supply provides a 5V voltage to the PIC board. Outputs on client side are observed using the oscilloscope. The interface between the server (FPGA) and the client (PIC) is provided by connecting the GPIO ports on the FPGA side with the connector ports on the PIC add-on card using wires. PIC is programmed by writing an embedded C program. This program is then configured for PIC18F4245 using MPLAB IDE.

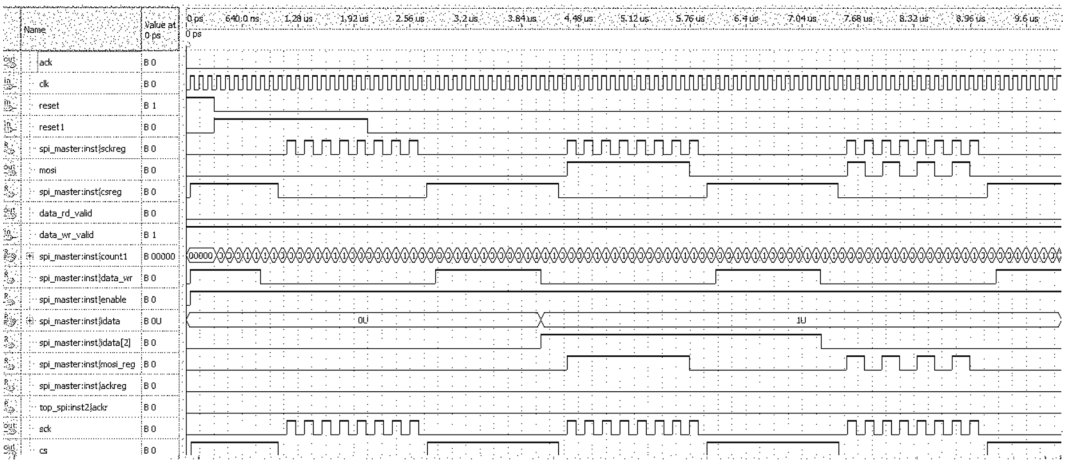

Figure 9: Timing simulation waveform for SPI.

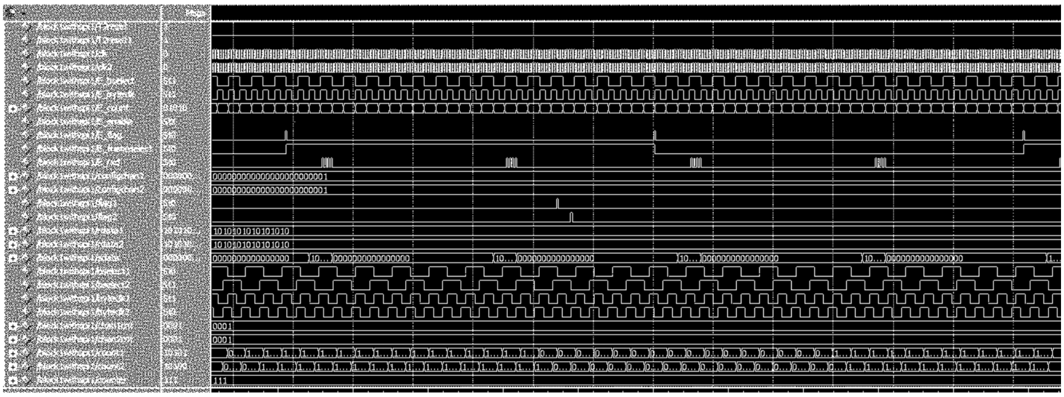

Figure 10: *Timing simulation waveform for T1/E1 channel serial interface.*

4. Simulation results

4.1. SPI (Serial Peripheral Interface)

Fig 9 shows timing simulation waveform for the SPI implementation. Initially, byte *00* is sent followed by byte *FE*. Byte *FE* indicates that the upcoming bytes after *FE* are the actual data bytes that need to be transferred. Table 1 indicates that the number of LUTs utilized by the design accounts to 39. The maximum frequency turns out to be 352.36 *MHz* which indicates that this technique may be utilized for very high speed requirements.

4.2. TDMA channel (T1/E1) serial interface

Fig 10 shows timing simulation waveform for T1/E1 serial interface. Table 2 indicates that the number of LUTs utilized by the design accounts to 78. Table 3 depicts the memory bits utilization by the dual port RAM used in the design.

Table 2: Resourc Usage Summary for T1/E1 channel interface.

Logic Elements	*Server*	*Client*	*interface*	*Total*
LUTs	13	0	14	27
Logic Registers	18	8	25	51

Table 3: Memory Utilization for T1/E1 channel interface.

Module	*Number of 8-bit memory locations*
Master	32
Slave	32

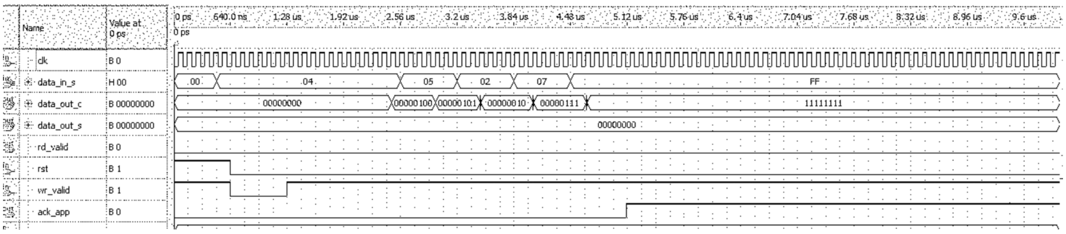

Figure 11: Timing simulation waveform for interface using FSM for master-transmit mode.

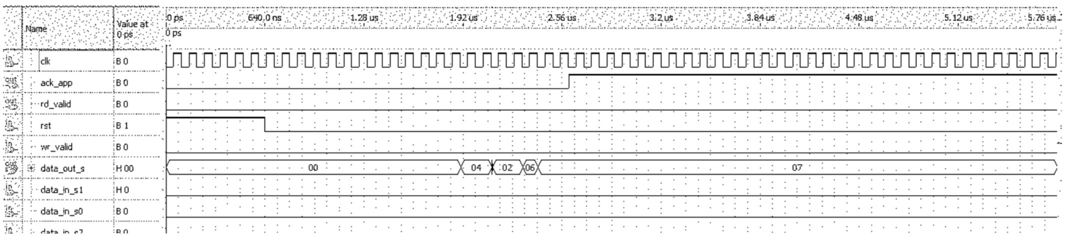

Figure 12: Timing simulation waveform for interface using FSM for master-receive mode.

Table 4: Resourc Usage Summary for interface using FSM.

Logic Elements	*Master*	*Slave*	*Total*
LUTs	120	78	198
Logic Registers	56	28	84

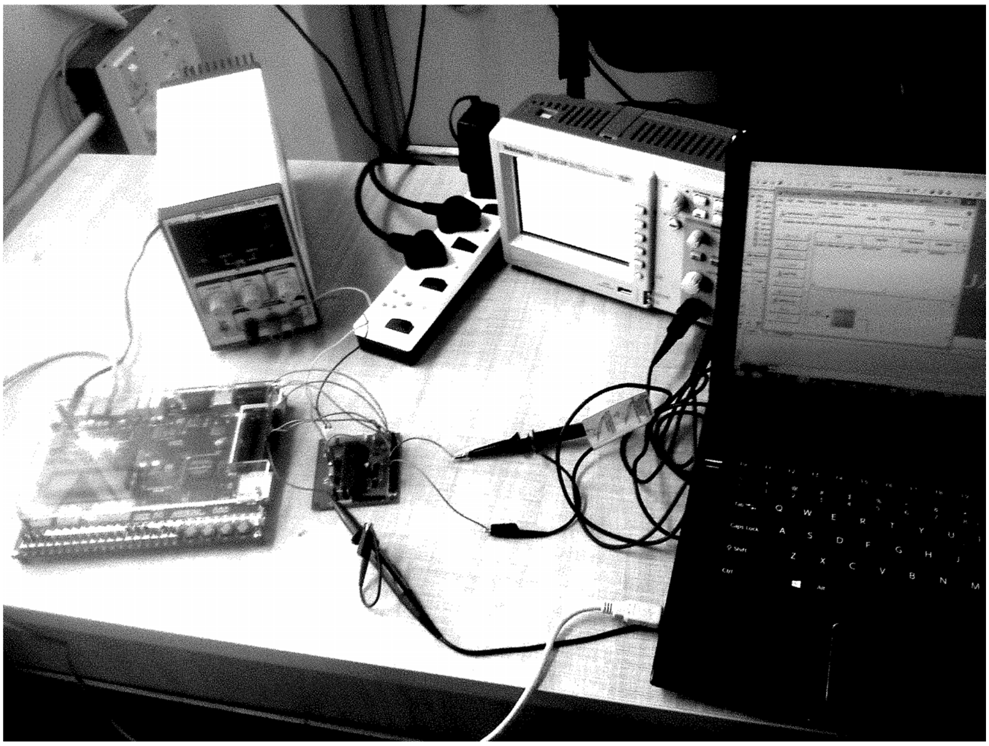

Figure 13: FPGA-microcontroller interface setup snapshot.

Table 5: Restricted Maximum Frequency for interface using FSM.

Interface	*Restricted FMAX for master clock*	*Restricted FMAX for slave clock*
SPI	352.36 *MHz*	-
T1/E1 channel interface	138.06 *MHz*	120.95 *MHz*
Interface using an FSM	103.37 *MHz*	-

4.3. Interface between FPGA and a bare microcontroller using a state machine

Altera Quartus II v12.1 is used for design of the FSMs. Static timing analysis is done for the design using Time Quest Timing Analyzer. The timing reports reveal that timing has been met for the complete design. As depicted by Table 5, the restricted maximum frequency (FMAX) is well above the range of frequency required for such an interface (typically 10 to 40 MHz). Timing simulation is carried out for all three cases namely, server-transmit/client-receive mode, server-receive/client-transmit mode and the conflicting case. Fig 11 and 12 show timing simulation results when simulation is done using Quartus Simulator (QSIM). Table 4 depicts the resource usage summary for server and client modules. The total number of LUTs utilized by the design are 282. Thus the area overhead introduced in this design is fairly negligible.

5. Conclusion

In this proposal, SPI, TDMA (T1/E1) channel serial interface and interface using an FSM have been designed. The techniques are implemented and tested on Altera Cyclone II FPGA present on Altera DE II (Development and Education) board and PIC18F4245 microcontroller. Results reveal that the frequencies (FMAX) are well above required frequency ranges for an FPGA-Microcontroller interface. Results also imply that the area overhead is negligible in all the three cases. Thus these cost effective interfacing strategies prove out to be highly reliable and may be used in integrating high end embedded applications.

References

[1] Conrad A. Theron, Donald H. St. Pierre, Jr., “Method and apparatus for changing execution code for a microcontroller on an FPGA interface device”, U.S. Patent 6631520 B1, May 14, 1999.

[2] Craig Nemecek, “Host to FPGA interface in an in-circuit emulation system”, U.S. Patent 7206733 B1, October 10, 2001.

[3] Warren Snyder, Craig Nemecek, Bert Sullam, “Emulator chip/board architecture and interface”, U.S. Patent 7076420 B1, July 11,2006.

[4] Warren Snyder, “Capturing test/emulation and enabling real-time debugging using an FPGA for in-circuit emulation”, U.S. Patent 7188063 B1, March 6, 2007.

[5] Jose Maria Insenser Farre, Julio Faura Enriquez, “Microprocessor based mixed signal field programmable integrated device and prototyping methodology”, U.S. Patent 6460172 B1, October 1, 2002.

[6] *www.alcatel-lucent.com*

[7] *www.bpl.in*

[8] *www.cisco.com*