FPGA Implementation of Smart Multi-Protocol Translator

Raju Patil, Pandit Nad, Smt.Sujatha Hiremath

Abstract --- The protocols are very important to achieve the interoperability between two devices. The commonly used protocols are SPI,GPIO,UART,I2C etc .These compact protocol converters create seamless, low-power, low-voltage connections for the most widely used interfaces (I2C, GPIO, IrDA, SPI, UART), making it easy to add multiple devices to any application with UART or SPI bus interfaces. The result is increased design flexibility with reduced complexity, lower software overhead, and faster time-to-market.In this paper we focus on the implementation of a new smart bus translator or protocol converter to convert standard protocol of one device to the protocol of other device [SPI, UART, and GPIO]. It reduces the complexity of circuit design, if we are having different protocol based devices it is very difficult to build separate platform and software/firmware for each device, in such case it is very useful. This design also reduces the firmware development time for different protocol based devices. For implementation was carried out using Xilinx ISE 12.3 Platform (software) and Spartan 3E FPGA Platform (Hardware).

Keywords--- SPI, UART, GPIO, Multi-protocol translator and Smart Bus Translator.

1 INTRODUCTION

In most of the applications, the physical systems require a real-time operation to interface high speed constraints. A Bus translator or Protocol Converter is a device used to convert standard or proprietary protocol of one device to the protocol suitable for the other device or tools to achieve the interoperability. The design intent is to develop a bridge that provides interoperability between different serial protocols based devices. A smart and configurable protocol bridge or bus translator serves as bus interpreter for peripheral interfaces with different protocols. There are in numerous devices out in the digital world serving various purposes with limited options for connectivity. For instance there are thousands of sensors, transducers, memory devices, controllers and other chips which provide interface via just one or two serial protocols. It is extremely difficult to develop common firmware/software for various such devices if they all operate similar as required but communicate with different protocols.

- ***Raju patil** , currently pursuing masters degree program in Vlsi Design & Embedded system in R V College of Engineering Bangalore,Karnataka, India, E-mail: rnbec021@gmail.com.*
- ***Pandit Nad**, currently pursuing masters degree program in Vlsi Design & Embedded system in R V College of Engineering Bangalore,Karnataka, India, E-mail:pandit.nad@gmail.com.*
- *Smt.Sujatha Hiremath ,Asst.professor,R V College of Engineering Bangalore,Karnataka,India , E-mail: sujathah@rvce.edu.in.*

Testing of such devices will result in an overhead work of development of circuitry, and firmware/software for each of such devices with different protocols. There is a necessity for a bridge that eliminates this problem. This solution can eliminate development time of firmware for similar devices with different protocols.

For example various families of GPS chips come with UART interface, some with I2C interface, and some with GPIO interface. It is a overhead to develop three different platforms and firmware/software for each of interfaces. A protocol bridge would come very helpful in such applications. Also a bridge helps in development of application despite of interface type and also provides the luxury of changing to a different device with different interface without having to make any changes to existing software and configurations.

F. Leens [6] has designed FPGA implementation of SPI and I2C on Spartan 3e.

A.K. Oudjida [3] has also designed FPGA implementation of SPI and I2C and done comparative study of both these protocols. All other reference papers are referred for the design of SPI, UART and GPIO protocols.

In all these papers discussed, the bus protocol translator translates only one protocol to another. And also they do not account for multiple protocol translation. This lacuna has been addressed in this paper by designing a multi-protocol translator, which translates protocols between SPI, UART, and GPIO.

The paper is organized as follows: section 2 describes the proposed design with three micro-architectures namely SPI, UART & GPIO; in section 3 syntheses and output results were plotted; section 4 discusses the conclusion and future work.

2 PROPOSED DESIGN

Design and implementation of Bus translator:
The Bridge has been designed with UART, SPI and GPIO interfaces. The design has two interfaces of each protocol type serving as a universal bridge between these protocols. Any one of these protocols could be converted to other and vice versa. We have to configure the communication path which we want by using config.select pin. The design will support operating configurations shown in table 1 and the proposed design shown in the figure 1.

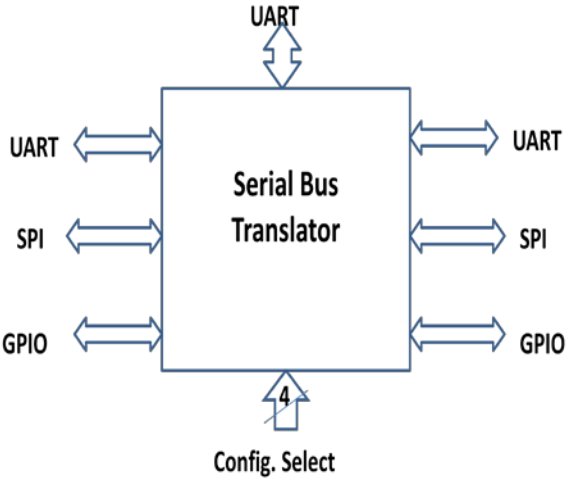


Figure1. Proposed Design of Smart Bus Translator

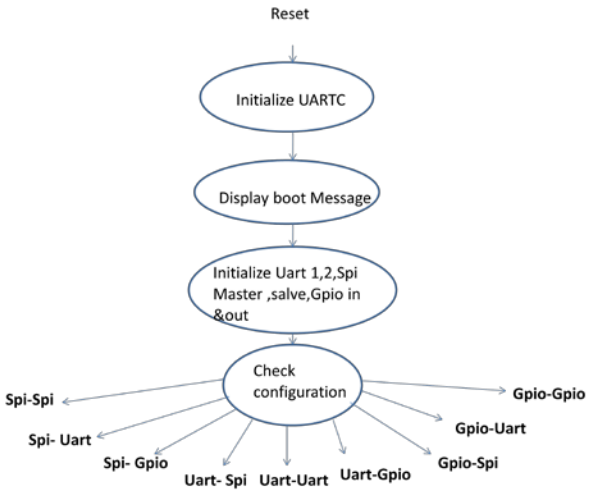


Figure 2.FSM of proposed design.

Table 1. Configuration of SBT

Configuration	Config .select	Additional Features
UART– UART	**0000**	Interrupt/req based data acquisition. Slave operation.
UART – SPI	**0001**	Delay insertion, data manipulation, extended slave selection.
UART– GPIO	**0010**	Slave operation.
SPI – UART	**0011**	Delay insertion, different baud rates on each interface.
SPI – SPI	**0100**	Slave selection, Master operation.
SPI – GPIO	**0101**	Interrupt based data acquisition.
GPIO – UART	**0110**	Interrupt based data acquisition.
GPIO – SPI	**0111**	Interrupt based master operation.
GPIO– GPIO	**1000**	Data manipulation, delay insertion.

Algorithm of proposed design [figure 2] shows the step wise execution of Protocol translator. After reset the device initialize the configuration Uart, and

display the boot message on the screen, after this it initialize all protocols in the proposed design, based on the configuration select, it provides the operation.

2.1 Architecture of SPI (Serial Peripheral Interface)

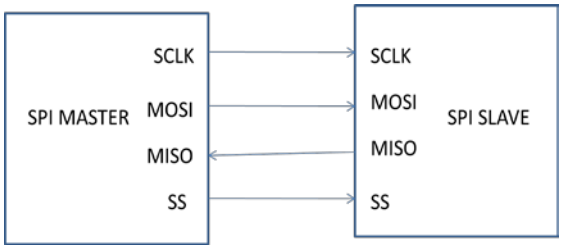
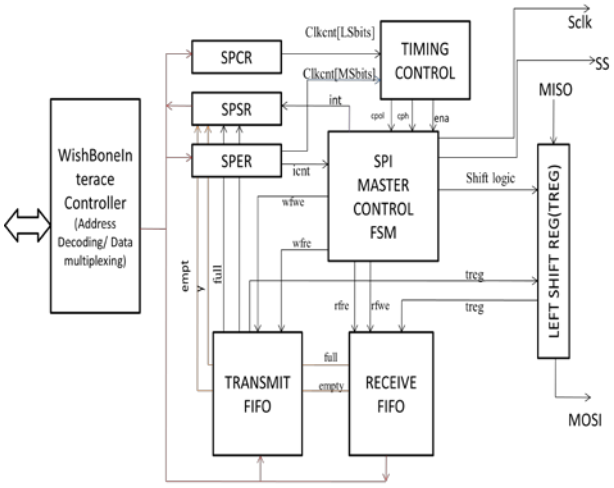


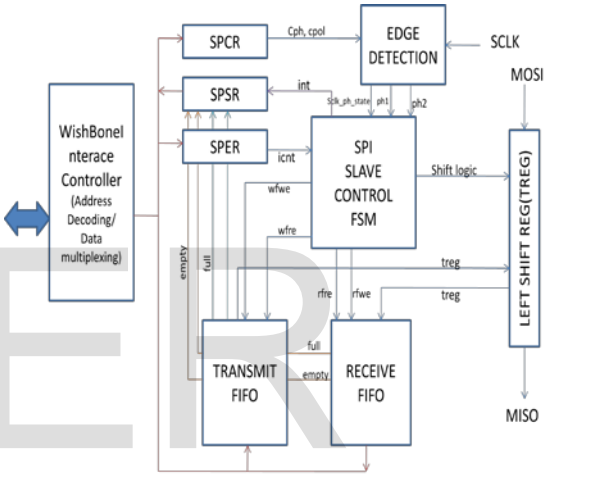
Figure 3. Architectural block diagram

Serial to Peripheral Interface (SPI) is a hardware/firmware communications protocol developed by Motorola and later adopted by others in the industry. Microware of National Semiconductor is same as SPI. Sometimes SPI is also called a "four wire" serial bus. 4-wire serial communications interface used by many microprocessor / microcontroller peripheral chips that enable the controllers and peripheral devices to communicate each other, it shown in the fig 3. The SPI bus, which operates at full duplex (means, signals carrying data can go in both directions simultaneously), is a synchronous type data link setup with a Master / Slave interface. Due to this high-speed aspect, the bus lines cannot be too long, because their reactance increases too much, and the Bus becomes unusable. However, it's possible to use the SPI Bus outside the PCB at low speeds, but this is not quite practical. 4-wire serial communications interface used by many microprocessor / microcontroller peripheral chips that enable the controllers and peripheral devices to communicate each other. The SPI bus, which operates at full duplex (means, signals carrying data can go in both directions simultaneously), is a synchronous type data link setup with a Master / Slave interface and can support up to 1 mega baud or 10Mbps of speed.

Micro architecture design of SPI:



(a)



(b)

Figure 4. SPI a) Master and b) Slave

Due to this high-speed aspect, the bus lines cannot be too long, because their reactance increases too much, and the Bus becomes unusable. However, it's possible to use the SPI Bus outside the PCB at low speeds, but this is not quite practical.

Data and control lines of the SPI and the basic connection:

An SPI protocol specifies 4 signal wires.

1. Master Out Slave In (MOSI) - MOSI signal is generated by Master, recipient is the Slave.

2. Master In Slave Out (MISO) - Slaves generate MISO signals and recipient is the Master.

3. Serial Clock (SCLK or SCK) - SCLK signal is generated by the Master to synchronize data transfers between the master and the slave.

4. Slave Select (SS) from master to Chip Select (CS) of slave - SS signal is generated by Master to select individual slave/peripheral devices. The SS/CS is an active low signal.

In single-master protocol, usually one SPI device acts as the SPI Master and controls the data flow by generating the clock signal (SCLK) and activating the slave it wants to communicate with slave-select signal (SS), then receives and or transmits data via the two data lines it is shown in fig 4. A master, usually the host micro controller, always provides clock signal to all devices on a bus whether it is selected or not.

2.2 Architecture of UART :(Universal Asynchronous Receiver & Transmitter)

A universal asynchronous receiver/transmitter, abbreviated as UART, is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTS are commonly used in conjunction with communication standards such as RS-232, RS-422 OR RS-485.

Micro architecture design (UART):

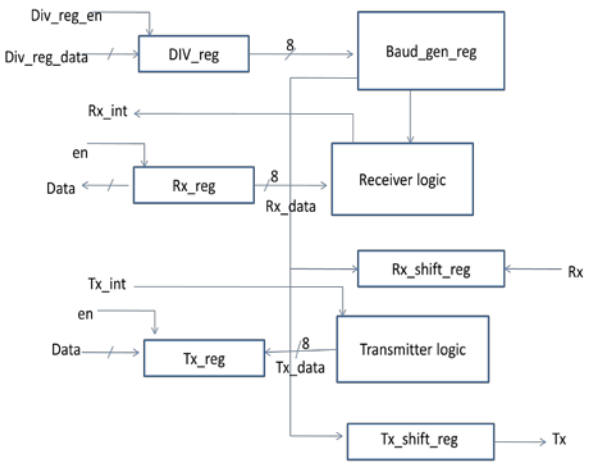


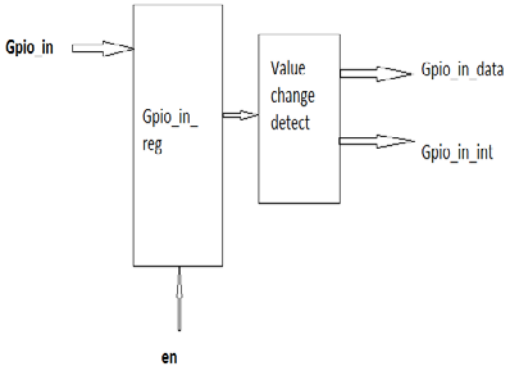
Figure 5. UART design

The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) Typically are handled by a special driver circuit external to the UART.

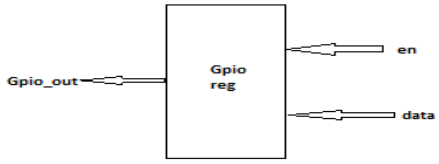
A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port [fig 5]. UARTS are now commonly included in microcontrollers. A dual UART, or DUART, combines two UARTS into a single chip. Many modern ics now come with a UART that can also communicate synchronously; these devices are called USARTS. The simple Universal Asynchronous Receiver Transmitter provides a reduced silicon footprint serial interface for an embedded system. This is ideal for a debug interface or simple console interface where the baud rate does not need to be changed by the application. The configurable FIFO depth ensures that the UART can be dimensioned for the application, and the software overhead of serial communication can be optimized without risking losing characters. The independent transmit and receive interrupts simplify the software architecture.

2.3 Architecture of GPIO (General Purpose Input & Output):

Micro architecture Design (GPIO):



(a)



(b)

Figure 6. GPIO a) Input and b) Output

General Purpose Input/output (GPIO) is a generic pin on a chip whose behavior (including whether it is an input or output pin) can be controlled (programmed) by the user at run time is shown in fig 6.GPIO pins have no special purpose defined, and go unused by default. The idea is that sometimes the system integrator building a full system that uses the chip might find it useful to have a handful of additional digital control lines, and having these available from the chip can save the hassle of having to arrange additional circuitry to provide them. GPIO peripherals vary quite widely. In some cases, they are very simple, a group of pins that can be switched as a group to either input or output. In others, each pin can be set up flexibly to accept or source different logic voltages, with configurable drive strengths and pull ups/downs. The input and output voltages are typically, though not universally limited to the supply voltage of the device with the GPIOs on, and may be damaged by greater voltages.

3 SYNTHESIS RESULTS

Fig 5 to 14 shows the output waveform and RTL view of the each protocol used in the proposed design.

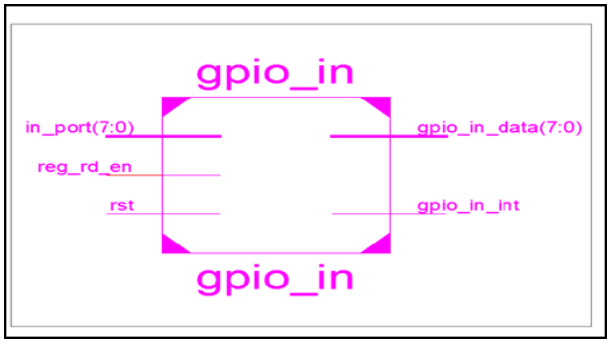


Fig 5.RTL view of GPIO input block

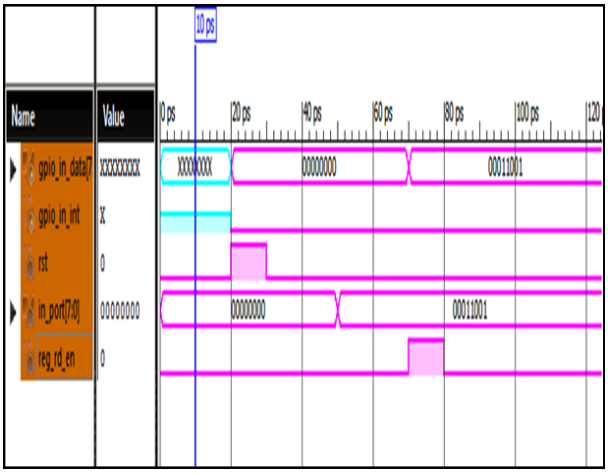


Fig 6. Output of GPIO input block

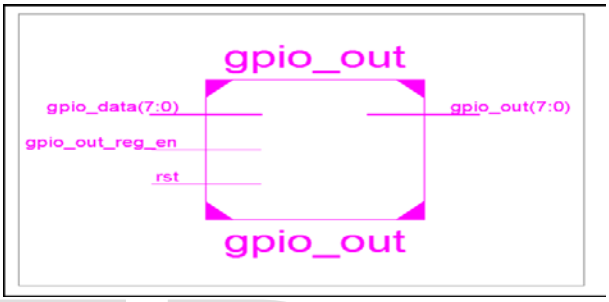


Fig 7. RTL view of GPIO output block

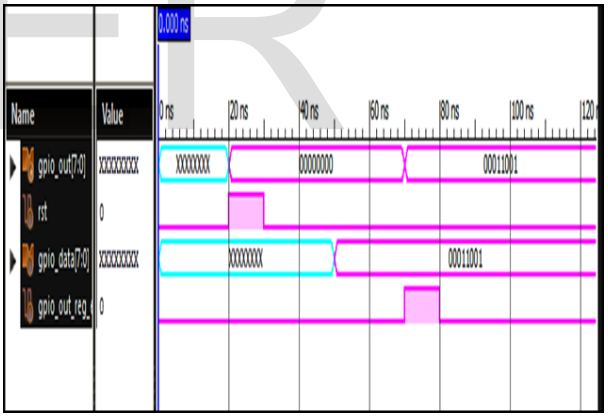


Fig 8. Output of GPIO output block

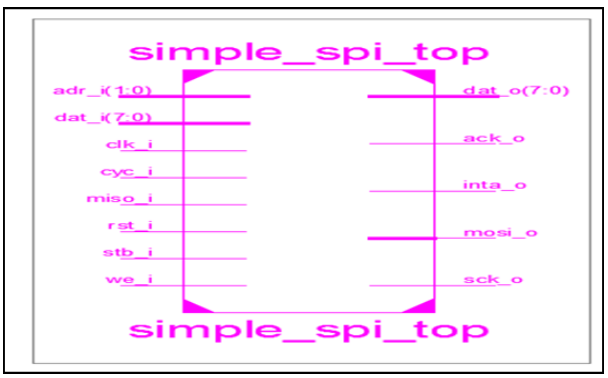


Fig 9. RTL view of Master block

Fig 10. Output waveform of SPI Master block

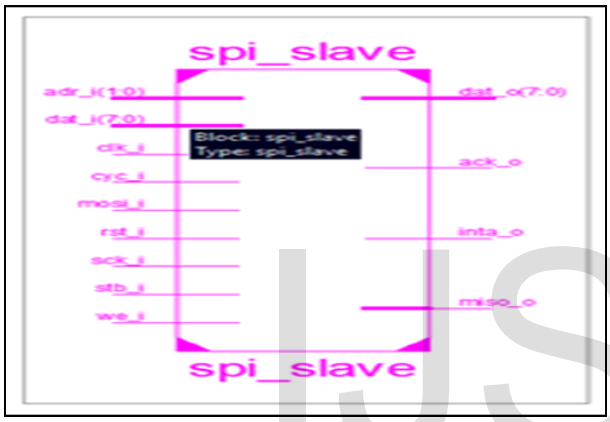


Fig 11. RTL view of slave block

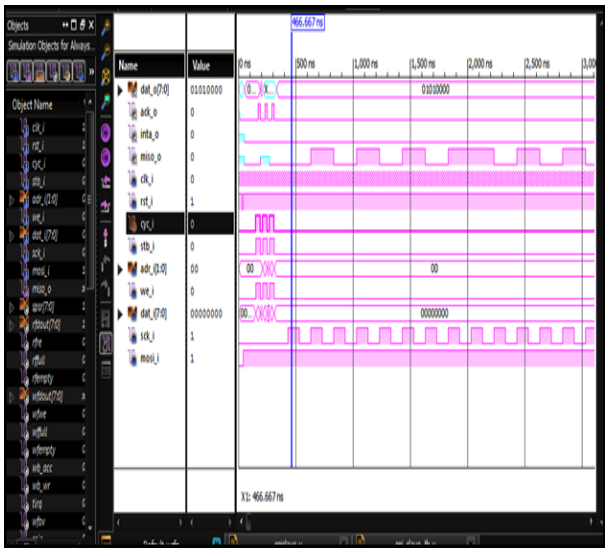


Fig 12. Output waveform of slave block

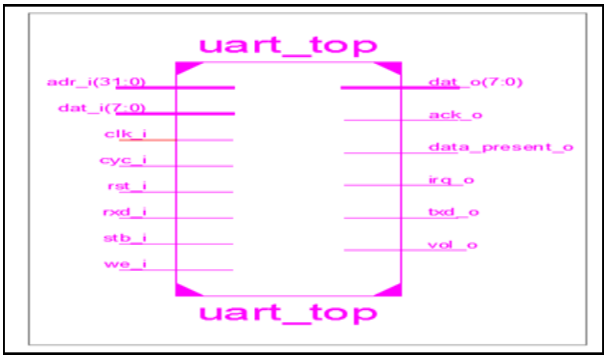


Fig 13. RTL view of UART block

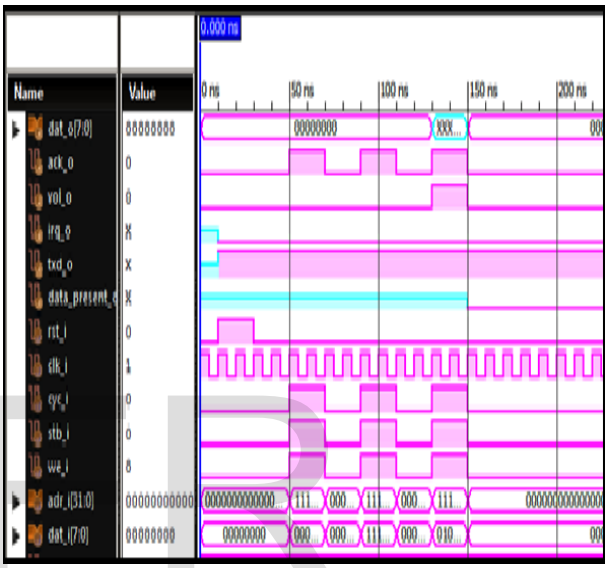


Fig 14. Output waveform of UART block

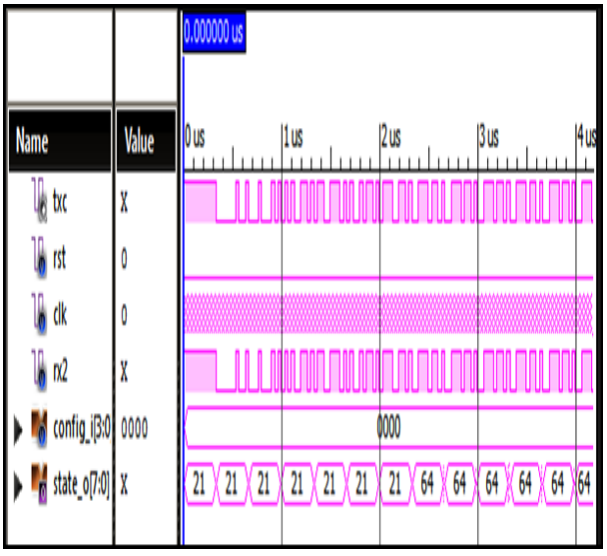


Fig 15.Output waveform of UART-UART configuration

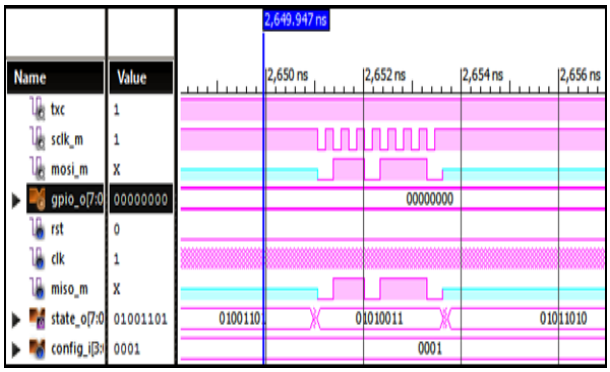


Fig 16.Output waveform of UART-SPI configuration

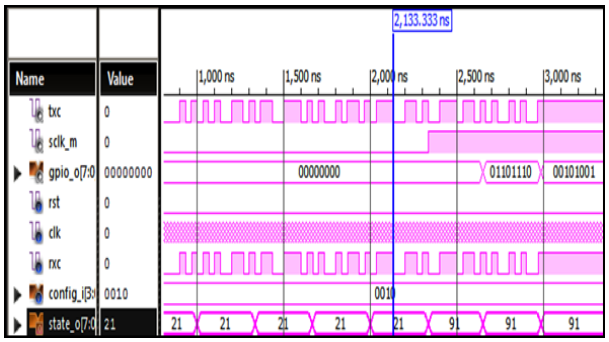


Fig 17.Output waveform of UART-GPIO configuration

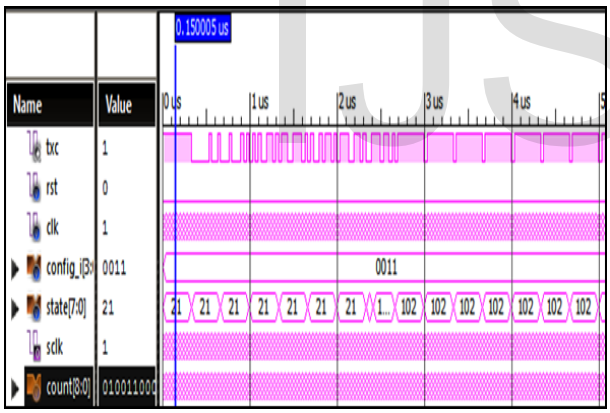


Fig 18.Output wave form of SPI-UART configuration

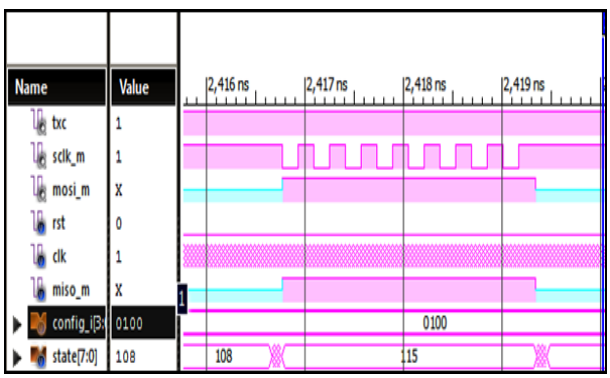


Fig 19.Output wave form of SPI-SPI configuration

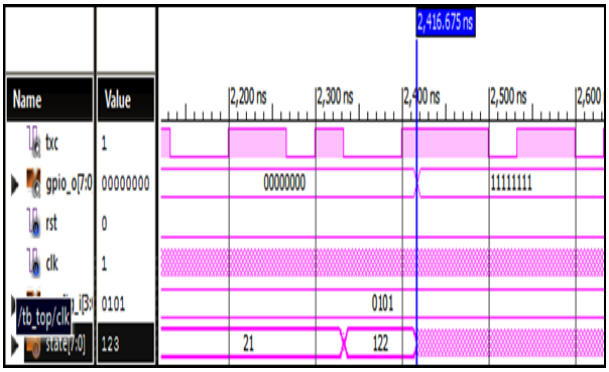


Fig 20.Output waveform of SPI-GPIO configuration

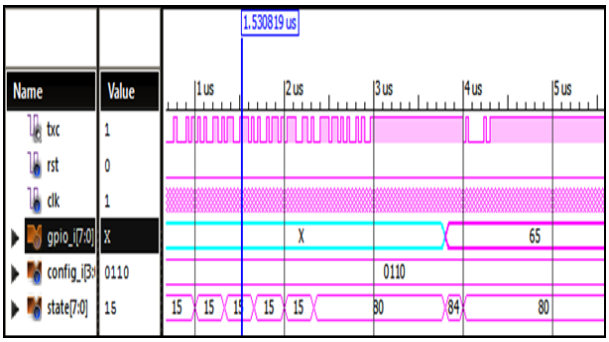


Fig 21.Output waveform of GPIO-UART configuration

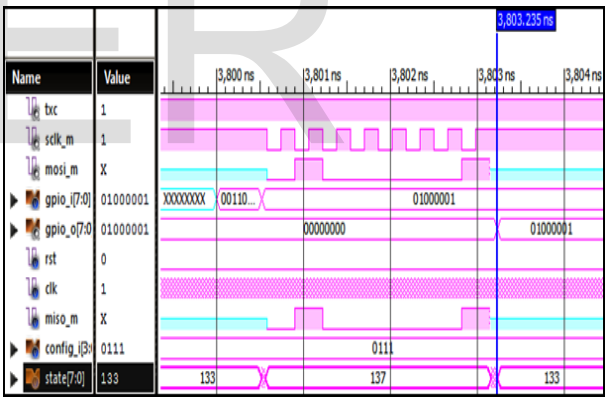


Fig 22.Output waveform of GPIO-SPI configuration

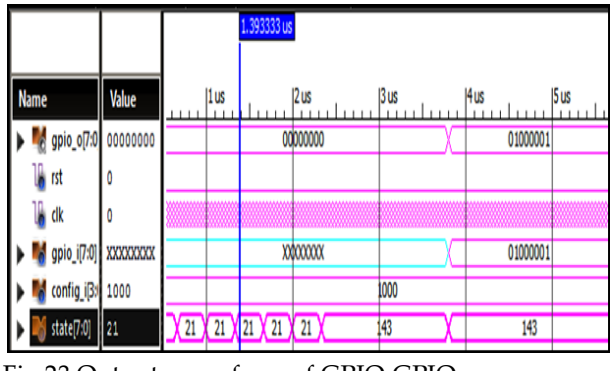


Fig 23.Output wave form of GPIO-GPIO configuration

4 CONCLUSION & FUTURE WORK

The paper intent was to design a new bus protocol translator. This translator can translate data between any of (UART, SPI, GPIO) protocols. The proposed design was carried out in verilog HDL .The design has been successfully simulated and functionally verified in ISE simulator. The hardware implementation has been done on Spartan 3E FPGA. The proposed design can be operated at different frequencies (up to 32MHz).This concept can be extended to include many more protocols that exist and also for new protocols to come.

REFERENCES

1. Free scale Semiconductor, (2008, October 14). Free scale SPI Block Guide V04.01 Jul.14
2. F. Leens, "An Introduction to SPI Protocols ,"IEEE Instrumentation & Measurement Magazine, pp. 8-13, February 2009.
3. A.K. Oudjidaetai, FPGA Implementation of I2C & SPI protocols. A Comparative Study". Proceedings of the 16th edition IEEE International Conference on Electronics Circuits and System ICECS, pp.507 -510, Dec 13-16 2009.
4. www.microsemi.com/soc/download/rsc/f=UART_to_SPI_DF.
5. J.M. Irazabel& S. Blozis, Philips Semiconductors, "I2C-Manual,"Application Note, ref. AN10216-0, March 24, 2003.
6. F.Leens, "An Introduction to I2C and SPI Protocols," IEEE Instrumentation & Measurement Magazine, pp. 8-13, February 2009.
7. Wolfson Microelectronics. (2004) "WM8731 Data sheet". PDF Document.
8. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, "They (protocols) are to communication what programming languages are to computation".
9. Ben-Ari 1982, chapter 2 - The concurrent programming abstraction, p. 18-19, states the same.
10. ALSA Development List, Linux, Linux TV GPIO Pins Info for Gpio protocol.
11. Design and simulation of UART serial communication module based on VHDL - Fang Yi-Yuan, Chen Xue, IEEE Explore, may 2011.
12. A.K Oudjida et ai, Master-Slave wrapper communication protocol: A case-study, Proceedings of the 1st IEEE International Computer Systems and Information Technology Conference ICSIT'05, PP 461-467, 19-21 July 2006.