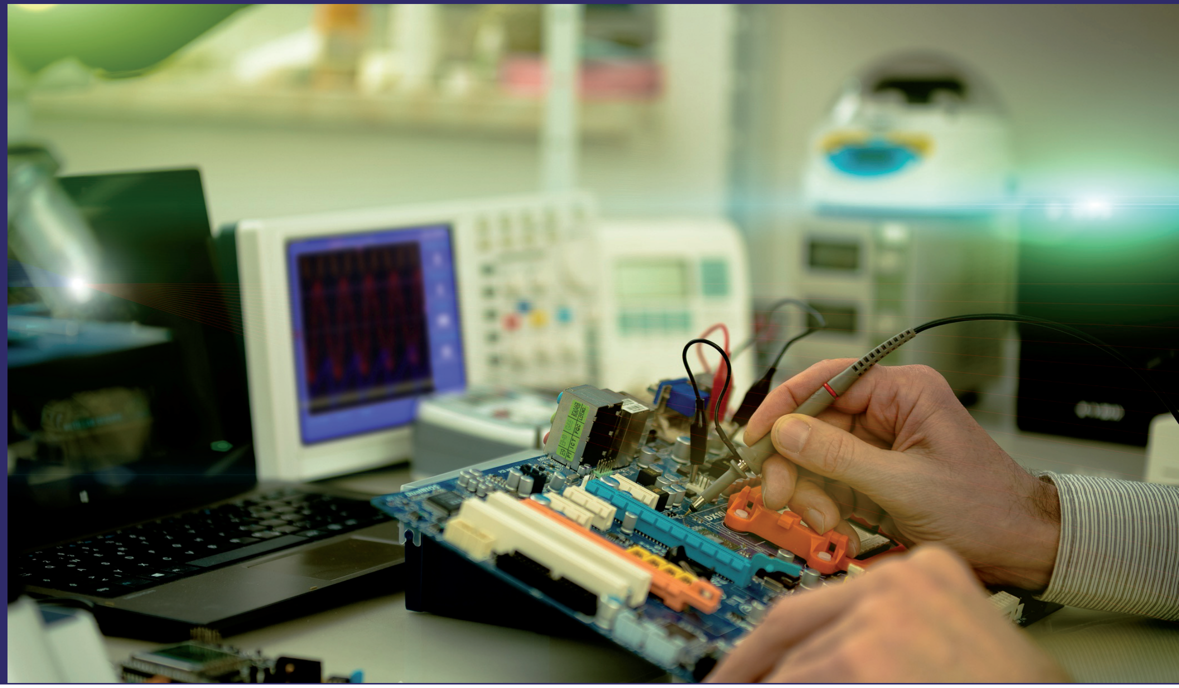
This book is about the use of embedded devices such as FPGA board , ARM boards on a shared network. It is important step to use FPGA to design and improve flow of multi user data in a high speed computer network environment. It will provide a more efficient way of utilizing a single board on a shared network where multiple users can acquire one board for their work but different time slot using TCP/IP



Sandeep Pansare
Satish Turkane
Chandrakant Bhos

Implementation Of Multi-user Fpga Environment Over TCP-IP

Prof Pansare Sandeep has completed his graduation in Electronics and telecommunication and post graduation in Embedded and VlSI. Currently he is working as assistant professor in Amrutvahini Coll. of Engg., Sangamner.

978-3-659-83260-4

LAP LAMBERT Academic Publishing

Sandeep Pansare
Satish Turkane
Chandrakant Bhos

Implementation Of Multi-user Fpga Environment Over TCP-IP

Sandeep Pansare
Satish Turkane
Chandrakant Bhos

Implementation Of Multi-user Fpga Environment Over TCP-IP

LAP LAMBERT Academic Publishing

Impressum / Imprint
Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
Alle in diesem Buch genannten Marken und Produktnamen unterliegen warenzeichen-, marken- oder patentrechtlichem Schutz bzw. sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Inhaber. Die Wiedergabe von Marken, Produktnamen, Gebrauchsnamen, Handelsnamen, Warenbezeichnungen u.s.w. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliographic information published by the Deutsche Nationalbibliothek: The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.
Any brand names and product names mentioned in this book are subject to trademark, brand or patent protection and are trademarks or registered trademarks of their respective holders. The use of brand names, product names, common names, trade names, product descriptions etc. even without a particular marking in this work is in no way to be construed to mean that such names may be regarded as unrestricted in respect of trademark and brand protection legislation and could thus be used by anyone.

Coverbild / Cover image: www.ingimage.com

Verlag / Publisher:
LAP LAMBERT Academic Publishing
ist ein Imprint der / is a trademark of
OmniScriptum GmbH & Co. KG
Bahnhofstraße 28, 66111 Saarbrücken, Deutschland / Germany
Email: info@lap-publishing.com

Herstellung: siehe letzte Seite /
Printed at: see last page
ISBN: 978-3-659-83260-4

Zugl. / Approved by: Pune, University Of Pune, Diss., 2012

Copyright © 2016 OmniScriptum GmbH & Co. KG
Alle Rechte vorbehalten. / All rights reserved. Saarbrücken 2016

ACKNOWLEGDEMENT

This work gives me an opportunity to express deep gratitude for the same. While preparing my dissertation report. I received endless help from number of people. This report would be incomplete if I don't convey sincere thanks to all those who were involved.

First and foremost, I thank my guide **Prof S. M. Turkane**, Head of Department, of Electronics and Telecommunication, PREC, Loni for giving me his valuable time and opportunity to present this dissertation report.

I give my sincere regards to **Prof. A. H. Ansari**, **Prof. S. A. Shaikh** and **Prof. S. G. Galande**, M. E. coordinator for their dispensable support, priceless suggestions and valuable time.

I also thank all other staff members for their advice and cooperation. Finally, I wish to thank my parents and for being supportive to me, without whom this seminar could not have seen light of the day.

Every work is outcome of full proof planning, continuous hard work and organized effort. This work is a combination of all three put together sincerely.

Mr. Pansare Sandeep Dilip

CONTENT

List of Figures

ABSTRACT

Optimizing the single target device as a simulator in a multi user environment is our aim to achieve. It is important step to use FPGA to design and improve flow of multi user data in a high speed computer network environment. Hence, we create a system where multiple users (computers) will use one target device (FPGA) through Ethernet without the need of actually carrying target device to each and every user. For this we are using a controller module which consist of control logic and user interface module.

It provides a more efficient way of utilizing a single board for hardware simulation. It could place itself in educational establishments where the prototyping board does not need to be changed from workstation to workstation and also provides an efficient way of managing expensive resources.

Chapter 1

Introduction

1.1 Project Overview

In this paper we look at a simple type of hardware simulation strategy where there is no altitude, orientation, no line of site limitation for simulating on the FPGA board and where multiple users can acquire the same board for their hardware simulation but at different time connecting through network over TCP/IP.

We will discuss hardware simulation, in particular for multi-user environment and different transmission media. We will also see the future possibilities to reduce the queue and accessibility of the FPGA board problem. A user can connect to FPGA device either by Ethernet, WI-Fi, SPI or internet. Out of these transmission medias we will concentrate on use of wired media.

1.2 Objectives

Optimizing the single FPGA board resource as a simulator in a multi user environment between end user and targeted device. In this project we look at a simple type of hardware simulation strategy where multiple users can acquire the same board for their hardware simulation but at different time connecting through TCP/IP. We explain the FPGA development platform we use for controller, which includes the TCP/IP interface through Ethernet for communication with user. We will also see the future possibilities to reduce the queue and accessibility of the FPGA board problem. A user can connect to target device either by Ethernet or WI-Fi / Wi-Max , internet . Out of these transmission Medias we will concentrate on use of Ethernet. We will use FPGA in both target device and controller.

The TCP/IP is an additional interface that is added to provide interface capabilities. The TCP/IP protocol uses a two-layer protocol: the higher layer TCP provides communication between the target board and source (PC) and breaks application layer information into packets and TCP/IP provides two methods of data delivery but our project aims at interface implemented for connection-orientated delivery using TCP.

IP is the protocol responsible for addressing and routing packets between networks. It ensures

they reach the correct destination network. IP deals with the physical network interface layer and for this we use Ethernet. The Ethernet module we designed to fit our requirements is based on a standard module provided by WIZNET W5100. The module contains an Ethernet port. TCP is the connection based communication method that will establish connection in advance and deliver the data through the connection by using IP Address and Port number of the systems. There are two methods to establish the connection. One is SERVER mode (passive open) that is waiting for connection request. The other is CLIENT mode (active open) that sends connection request to a server. The Client program has two versions. The Server program must be run on the machine which has the FPGA board plugged into an ISA slot. The Server creates a socket, through which it accepts the Clients connections. The socket is handled as a text file. Both the Server and Client can write and read from it. The Server contains the code for downloading the configuration.

The Client program must be run on the station that stores the configuration file. The Client needs two parameters: the name of the file containing the configuration to be downloaded and the IP address of the Server. The Client creates a socket too and tries to connect its socket to the server. If it succeeds, the Client sends a header, which contains the name of the file containing the configuration, and some commands to the Server. After the header, it sends the configuration data. The socket is handled as a text file and the configuration data is stored in the memory in binary format (independently from the file format), the Client must encode the binary data into textual data (like MIME). This is simple: the Client reads three bytes, which is $3 \times 8 = 24$ bits. It divides in four sections ($24 = 4 \times 6$ bits), so now he has four 6-bit values.

Chapter 2

Literature Survey

2.1 Existing Technology and Research

The ability to reconfigure an FPGA via the Internet was initially put forward by Fallside [1] and since then has been widely developed and commercially realized. Other work that is not directly related includes task management on FPGAs by Brebner [2] where such an idea could be applied to multiple users/processes sharing a FPGA and platform board for functional verification. This paper looks at a higher, simpler abstract view of efficiently utilizing an FPGA prototyping board. Such a strategy would bring benefits to FPGA rapid prototyping engineers, engineers changing workstations or on the move, and distributed design teams posted around the globe. It could also place itself in educational establishments where the prototyping board does not need to be changed from workstation to workstation and also provides an efficient way of managing expensive FPGA resources.

The feasibility of run-time reconfiguration of FPGAs has been established by a large number of case studies. However, these systems have typically involved an ad hoc combination of hardware and software. The software that manages the dynamic reconfiguration is typically specialized to one application and one hardware configuration. Dynamic reconfiguration of FPGAs has become viable with the introduction of devices that allow high speed partial reconfiguration, e.g., the Xilinx XC6200 series [3]. Dynamic reconfiguration is usually performed by a software system that decides when to reprogram part of the FPGA and with what. The simplest kind of run-time software simply selects a precompiled circuit and transmits the programming data directly to the FPGA.

Alternative to the use of Internet there has been also tremendous research in field of hardware-software co-synthesis of reconfigurable FPGA's. Hardware-software cosynthesis algorithms automatically produce hardware-software architectures for distributed embedded systems. Ideally, they minimize multiple costs, such as execution time, price, and average power consumption. Given a specification, a hardware-software co synthesis algorithm must select different processing elements (PEs) and communication resources to use in the embedded system (allocation), determine which resource will be used to carry outreach portion of the specification's computation and communication (assignment) and produce a schedule for all of the specification's computation and communication (scheduling). Given an embedded system specification, a co-synthesis algorithm produces a detailed description of an architecture that meets the design constraints and optimizes a set of costs.FPGAs are commonly used in distributed embedded systems. They share many traits with application-specific integrated circuits (ASICs); they are parallel hardware platforms. However, they have the advantages of reducing design time and supporting dynamic (runtime) reconfiguration. Although FPGAs typically have lower performance, higher power consumption, and higher energy consumption when compared with ASICs, for many applications, they have substantially better performance, average power consumption, and energy than general-purpose processors [4]. When a design uses dynamic reconfiguration, it is important to minimize the overhead, i.e., time and energy consumption, associated with reconfiguration. To reduce this overhead, much new reconfigurable architecture has been proposed [5]. In modern dynamically reconfigurable FPGAs, the embedded configuration storage circuitry can be updated selectively in a few clock cycles without disturbing the execution of the remaining logic. These new designs have increased the potential benefit of using dynamically reconfigurable FPGAs in low-power embedded systems by dramatically reducing the performance and energy penalties of dynamic reconfiguration. However, these costs are still substantial.

The increasing demand and use of computers in universities and research labs in the late 1960s generated the need to provide high-speed interconnections between computer systems. Ethernet was developed at Xerox PARC in 1973–1975. In 1976, after the system was deployed at PARC, Metcalfe and Boggs published a seminal paper, "Ethernet: Distributed Packet-Switching for Local Computer Networks."It is important step for IP and system development to use FPGA and IP core to design and improve flow of multi user data in a high speed communication internet environment and optimal utilization of available bandwidth. A user can connect to FPGA device either by Ethernet, WI-Fi, ISA or internet[6][7]. Out of these transmission medias we will concentrate on the use of wired media. There is no need for end users to have their own FPGA device. Our aim is to use Xilinx's system generator where multi users will use same IP Core over a distance sharing only one FPGA resource through internet.

Chapter 3

System Overview

3.1 Block Diagram

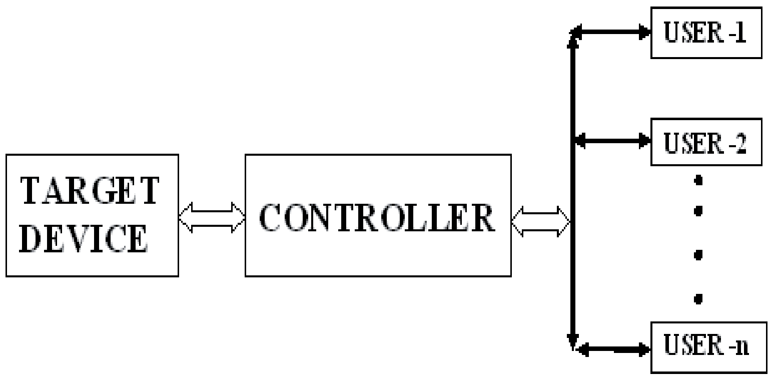


Figure: 3.1 System Block Diagram

We will see all these blocks in detail but first we will have an overview.

Target Device:

Target device is the device which will be used for downloading the user program. We will have such one target device in the network. This device will be shared in the network.

Controller:

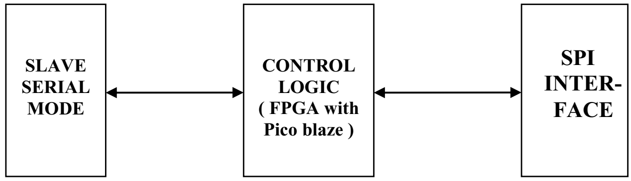


Figure: 3.2 Controller Architecture

In this we will have FPGA kit which will be loaded with Pico-Blaze is used for controlling purpose (controlling of target device) and hence called controller board. Slave serial mode is used to connect to target device while SPI interface is used for interfacing Ethernet module.

3.2 Target Device

Our aim is to share target device in a network. For this we have taken our target device as FPGA (SPARTAN 3E). Spartan-3E FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. The FPGA's configuration data is stored externally in a PROM or some other non-volatile medium, either on or off the board. After applying power, the configuration data is written to the FPGA using any of seven different modes: master Serial from a Xilinx Platform Flash PROM, Serial Peripheral Interface (SPI) from an industry-standard SPI serial Flash, Byte Peripheral Interface (BPI) Up or Down from an industry-standard x8 or x8/x16 parallel NOR Flash, Slave Serial typically downloaded from a processor, Slave Parallel, typically downloaded from a processor, JTAG, typically downloaded from a processor or system tester. Furthermore, Spartan-3E FPGAs support Multi boot configuration, allowing two or more FPGA configuration bit streams to be stored in a single parallel NOR Flash. The FPGA application controls which configuration to load next and when to load it. We will be using slave serial mode for our target device.

3.3 Controller

We are going to use soft processor core "PicoBlaze" an open source from Xilinx as our external host .This PicoBlaze soft processor core will be implemented in FPGA(control logic). The PicoBlaze microcontroller is a compact, capable, and cost-effective fully embedded 8-bit RISC microcontroller core optimized for the Xilinx FPGA families. The PicoBlaze microcontroller core is totally embedded within the FPGA and requires no external resources. The PicoBlaze microcontroller is extremely flexible. The basic functionality is easily extended and enhanced by connecting additional FPGA logic to the microcontroller's input and output ports. The PicoBlaze peripheral set can be customized to meet the specific features, function, and cost requirements of the target application. As the PicoBlaze microcontroller is delivered as synthesizable VHDL source code, the core is future-proof and can be migrated to future FPGA architectures, effectively eliminating product obsolescence fears. Being integrated within the FPGA, the PicoBlaze microcontroller reduces board space, design cost, and inventory.

The PicoBlaze microcontroller is specifically designed and optimized for the Spartan-3 family and with the support of Spartan-6 and Virtex-6 FPGA architectures. As it is delivered as VHDL source, the PicoBlaze microcontroller is immune to product obsolescence as the microcontroller can be retargeted to future generations of Xilinx FPGAs, exploiting future cost reductions and feature enhancements. Before the advent of the PicoBlaze and MicroBlaze embedded processors, the microcontroller resided externally to the FPGA, limiting the connectivity to other FPGA functions and restricting overall interface performance. By contrast, the PicoBlaze microcontroller is fully embedded in the FPGA with flexible, extensive on-chip connectivity to other FPGA resources. Signals remain within the FPGA.

The PicoBlaze microcontroller reduces system cost because it is a single-chip solution, integrated within the FPGA and sometimes only occupying leftover FPGA resources. The PicoBlaze microcontroller is resource efficient. Consequently complex applications are sometimes best portioned across multiple PicoBlaze microcontrollers with each controller implementing a particular function, for example, keyboard and display control, or system management. Microcontrollers and FPGAs both successfully implement practically any digital logic function. The program memory requirements grow with increasing complexity. Programming control sequences or state machines in assembly code is often easier than creating similar structures in FPGA logic. Microcontrollers are typically limited by performance. Each instruction executes sequentially, as an application increases in complexity, the number of instructions required to implement the application grows and system performance decreases accordingly. A microcontroller embedded within the FPGA provides the best of both worlds.

3.4 Ethernet Controller

We are using W5100 as ehernet controller. It is a full-featured, single-chip Internet-enabled 10/100 Mbps Ethernet controller designed for embedded applications where ease of integration, stability, performance, area and system cost control are required. It has been designed to facilitate easy implementation of Internet connectivity without OS . It is IEEE 802.3 10BASE-T and 802.3u 100BASE-TX compliant. Fig.4 shows the block diagram for our ehernet controller. The W5100 includes fully hardwired, market-proven TCP/IP stack and integrated Ethernet MAC & PHY. Hardwired TCP/IP stack supports TCP, UDP, IPv4, ICMP, ARP, IGMP and PPPoE which has been proven in various applications for several years. 16Kbytes internal buffer is included for data transmission. No need of consideration for handling Ethernet Controller, but simple socket programming is required. For easy integration, three different interfaces like memory access way, called direct, indirect bus and SPI, are supported on the MCU side. TCP is the connection based communication method that will establish connection in advance and deliver the data through the connection by using IP Address and Port number of the systems. There are two methods to establish the connection. One is SERVER mode(passive open) that is waiting for connection request. The other is CLIENT mode (active open) that sends connection request to a server.

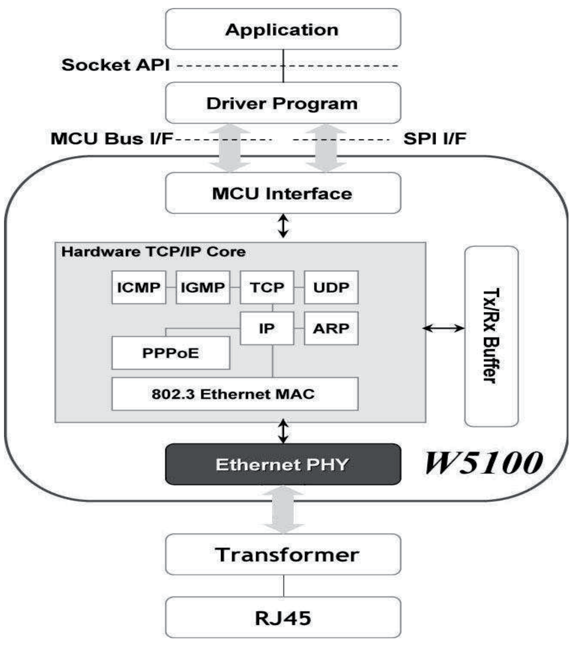


Figure: 3.3 Block diagram of WIZNET

Process of using General SPI Master Device:

1. Configure Input/output direction on SPI Master Device pins.
   /SS (Slave Select): Output pin
   SCLK (Serial Clock): Output pin
   MOSI (Master Out Slave In): Output pin
   MISO (Master In Slave Out): Input pin
2. Configure /SS as 'High'
3. Configure the registers on SPI Master Device.
   SPI Enable bit on SPCR register (SPI Control Register)
   Master/Slave select bit on SPCR register
   SPI Mode bit on SPCR register
   SPI data rate bit on SPCR register and SPSR register
   (SPI State Register)
4. Write desired value for transmission on SPDR register (SPI Data Register).
5. Configure /SS as 'Low' (data transfer start)
6. Wait for reception complete
7. If all data transmission ends, configure /SS as 'High'

3.5 Slave Serial Mode

In Slave Serial mode, an external host such as microcontroller writes serial configuration data into the FPGA (target device), using the synchronous serial interface. In Slave Serial mode (M[2:0] = <1:1:1>), an external host such as a microprocessor or Microcontroller writes serial configuration data into the FPGA, using the synchronous serial interface shown in Figure 3.4. The figure shows optional components in gray and uses a circled letter to associate a signal with more information found in the text. The serial configuration data is presented on the FPGA's DIN input pin with sufficient setup time before each rising edge of the externally generated CCLK clock input.

The intelligent host starts the configuration process by pulsing PROG_B and monitoring that the INIT_B pin goes High, indicating that the FPGA is ready to receive its first data.The host then continues supplying data and clock signals until either the DONE pin goes High, indicating a successful configuration, or until the INIT_B pin goes Low, indicating a configuration error. The configuration process requires more clock cycles than indicated from the configuration file size. Additional clocks are required during the FPGA's start-up sequence, especially if the FPGA is programmed to wait for selected Digital Clock Managers (DCMs) to lock to their respective clock inputs

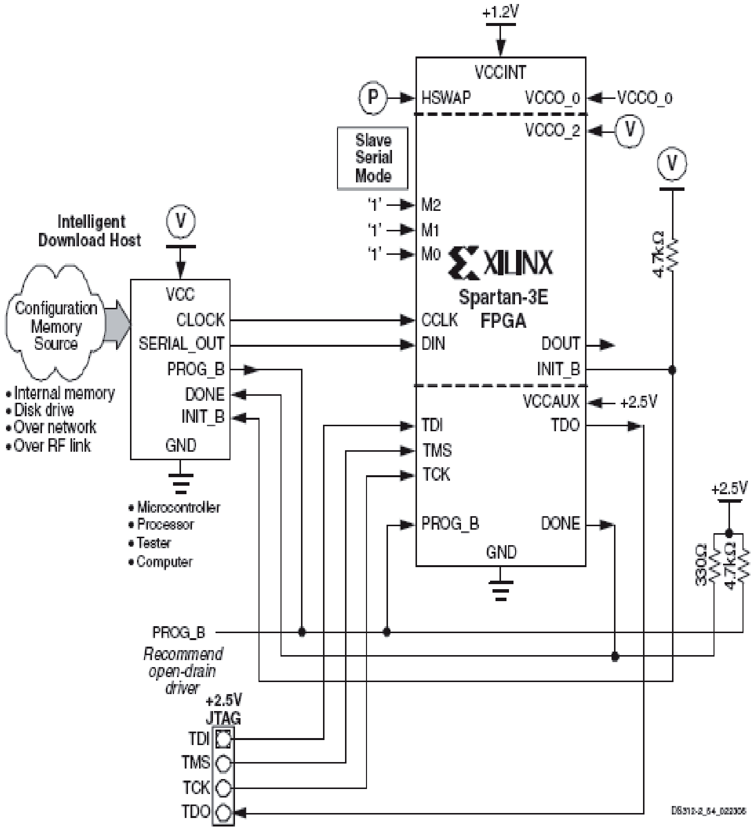


Figure: 3.4 Slave Serial Mode

Chapter 4

Operations

4.1 DEVICE OPERATIONS

The W5100 is controlled by a set of instruction that is sent from a host controller, commonly referred to as the SPI Master. The SPI Master communicates with W5100 via the SPI bus which is composed of four signal lines: Slave Select (/SS), Serial Clock (SCLK), MOSI (Master out Slave In), MISO (Master in Slave Out). The SPI protocol defines four modes for its operation (Mode 0, 1, 2, 3). Each mode differs according to the SCLK polarity and phase - how the polarity and phase control the flow of data on the SPI bus. The W5100 operates as SPI Slave device and supports the most common modes - SPI Mode 0 and 3.The only difference between SPI Mode 0 and 3 is the polarity of the SCLK signal at the in active state. With SPI Mode 0 and 3, data is always latched in on the rising edge of SCLK and always output on the falling edge of SCLK.

There are only two data lines used between SPI devices. So, it is necessary to define OP-Code. W5100 uses two types of OP-Code - Read OP-Code and Write OP-Code. Except for those two OP-Codes, W5100 will be ignored and no operation will be started. In SPI Mode, W5100 operates in “unit of 32-bit stream”. The unit of 32-bit stream is composed of 1 byte OP-Code Field, 2 bytes Address Field and 1 byte data Field. OP-Code, Address and data bytes are transferred with the most significant bit (MSB) first and least significant bit(LSB) last. In other words, the first bit of SPI data is MSB of OP-Code Field and the last bit of SPI data is LSB of Data-Field.

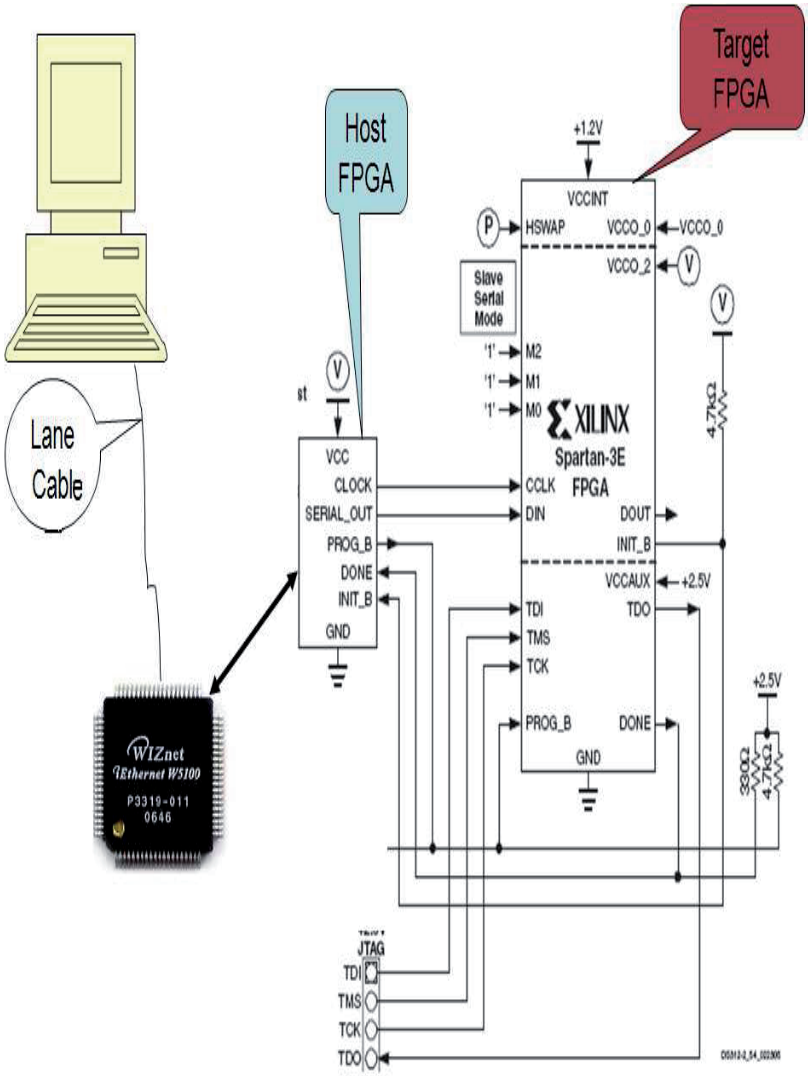


Figure: 4.1 Device Operations

Chapter 5

Layout Details

5.1 Top layout

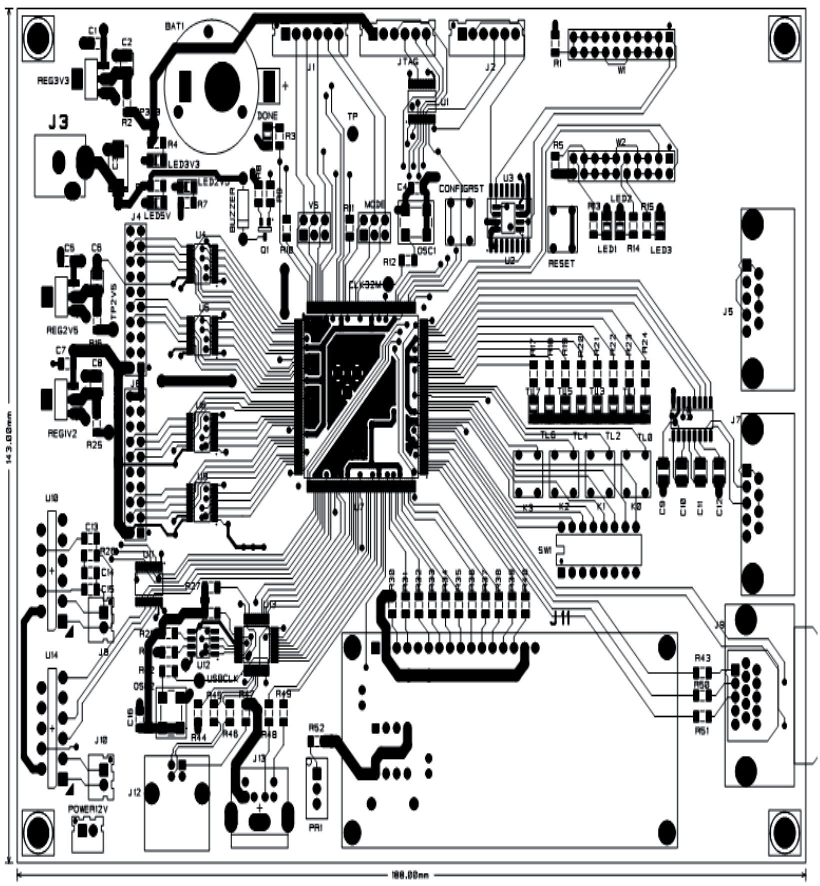


Figure: 5.1 Top layout

5.2 Bottom Layout

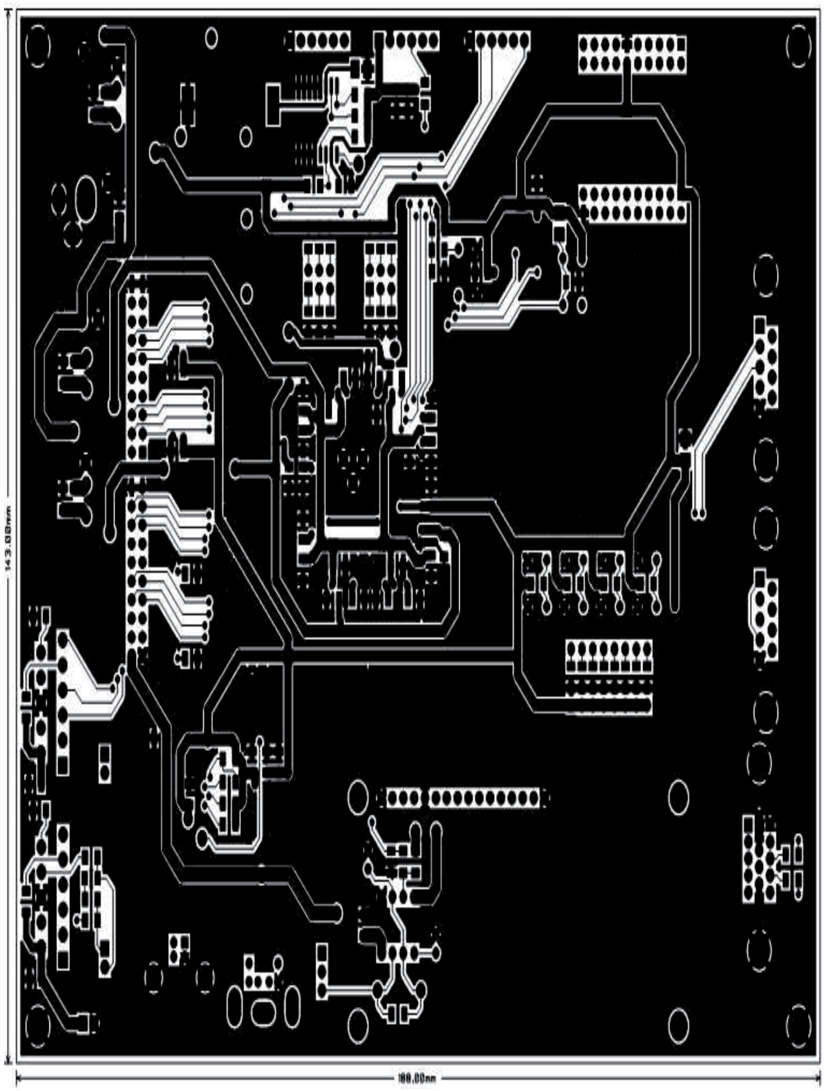


Figure: 5.2 Bottom Layout

Chapter 6

Hardware Design

6.1 Power Supply

In our project we have used three different voltages .These are 1.2V, 2.5V and 3.3V.

6.1.1 Power Jack

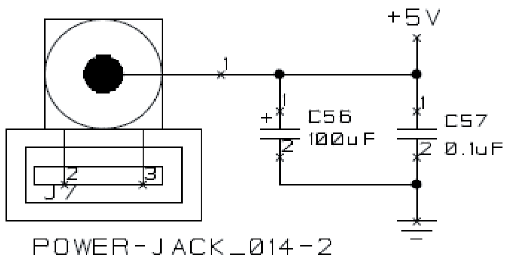


Figure: 6.1 Power Jack

6.1.2 1.2V Supply

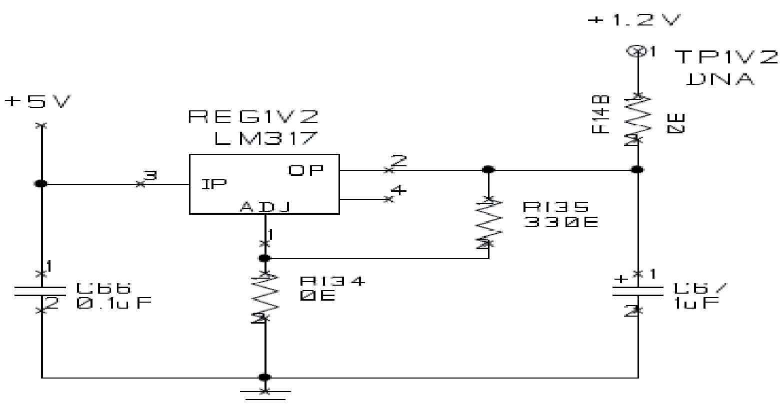


Figure: 6.2 1.2V Supply

6.1.3 2.5V Supply

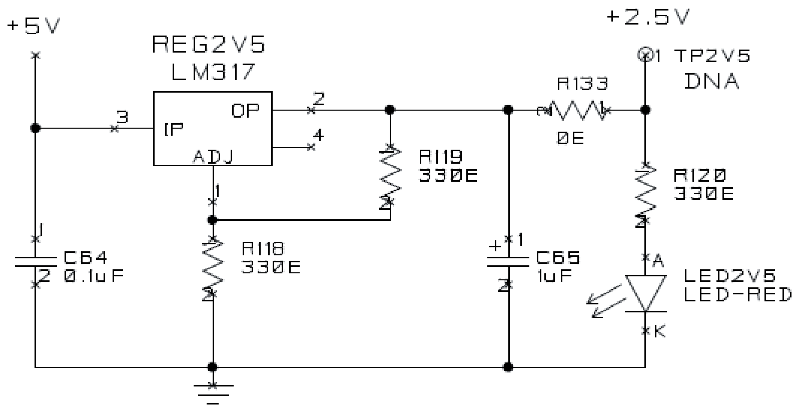


Figure: 6.3 2.5V Supply

6.1.4 3.3V Supply

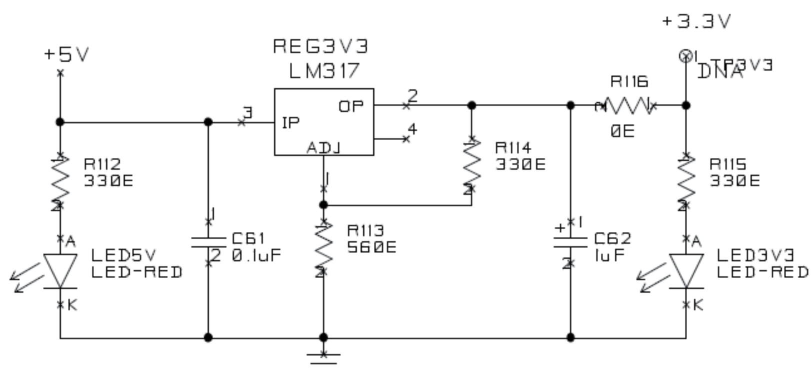


Figure: 6.4 3.3V Supply

6.2 PROM

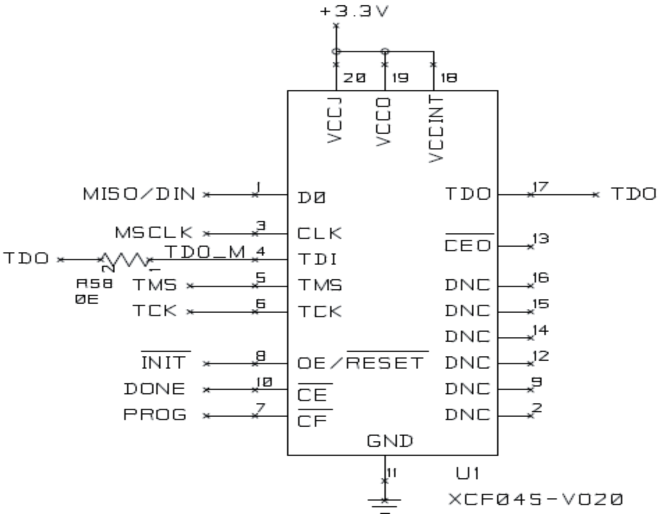


Figure: 6.5 PROM

We will be using 4MB PROM.

6.3 RESETS

6.3.1 Reset

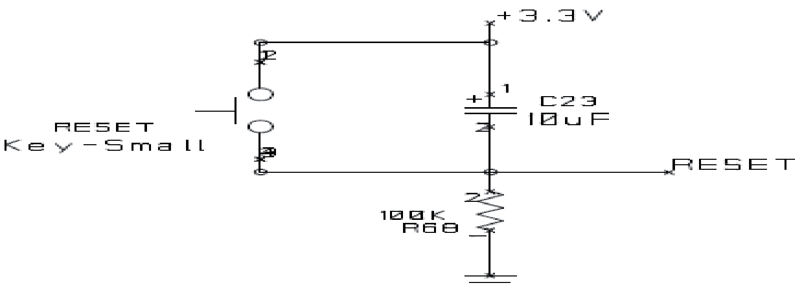


Figure: 6.6 Reset

This reset is located on target board. This reset is utilized for the target board.

6.3.2 Config Reset

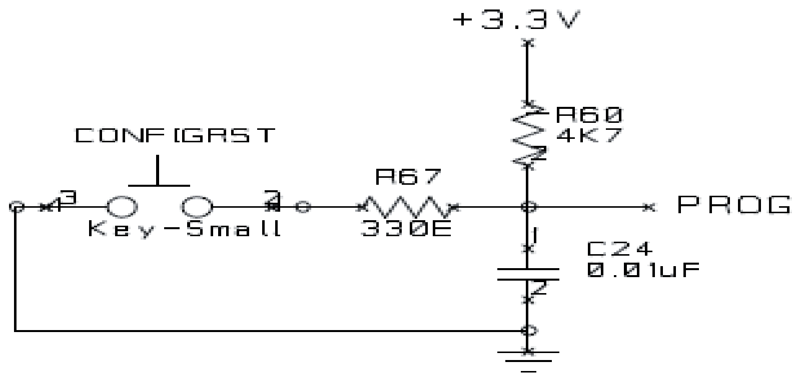


Figure: 6.7 Config Reset

Config reset is used on controller board. It is used when a new program is needed to be downloaded in target FPGA board.

6.4 Clock Circuit

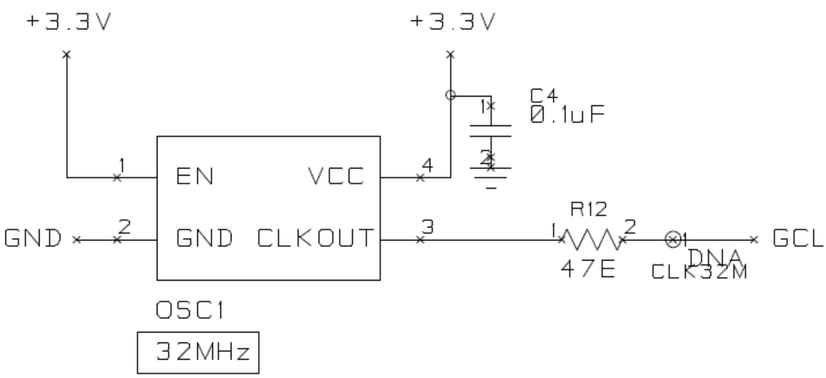


Figure: 6.8 Clock Circuit

6.5 SWITCHES

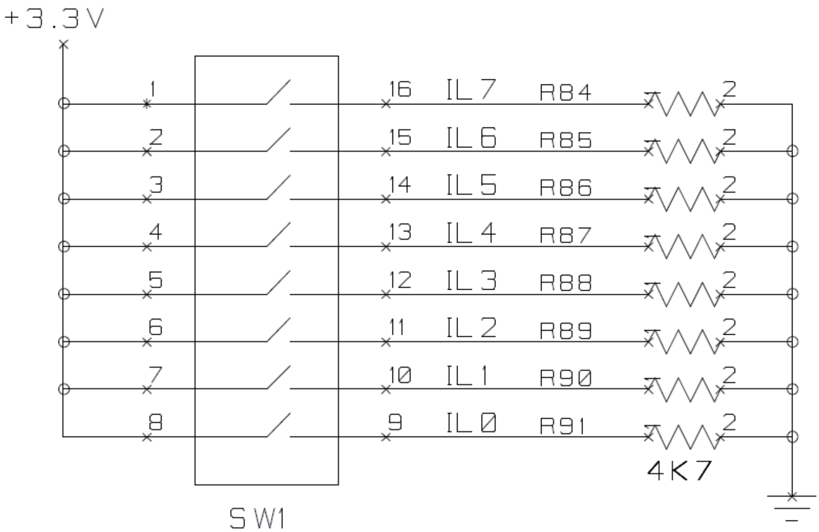


Figure: 6.9 Switches

6.6 LED

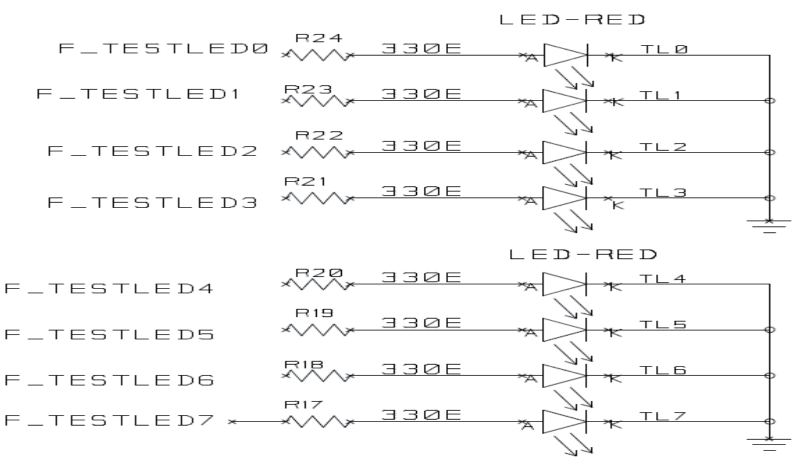


Figure: 6.10 LED

6.7 RS232

RS232 Standards

To allow compatibility among data communication equipment made by various manufactures, An interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. In 1963 it was modified and called RS232A. RS232B and RS232C were issued in 1965 and 1969, respectively. Here we refer it simply as RS232. Today, RS232 is the most widely used serial I/O interfacing standard. This standard is used on PCs and numerous types of equipment. However, since the standard was set long before the advent of TTL login family, its input and output voltage levels are not TTL compatible. In RS232, a 1 is represented by -3 to -25 V, while a 0 bit is +3V to +25V, making -3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converter such as MAX3232 to convert the TTL logic levels to the RS232 voltage level, and vice versa. MAX3232 IC chip are commonly referred to as line drivers.

RS232 pins

IBM introduced the DB-9 version of the serial I/O standard, which uses 9 pins only. The DB-9 pins are as shown below.

Pin	Description
1.	Data carrier detect (DCD)
2.	Receiver data (RxD)
3.	Transmitted data(TxD)
4.	Data terminal ready(DTR)
5.	Signal Ground(GND)
6.	Data set ready(DSR)
7.	Request to send(RTS)
8.	Clear to send(CTS)
9.	Ring indicator(RI)

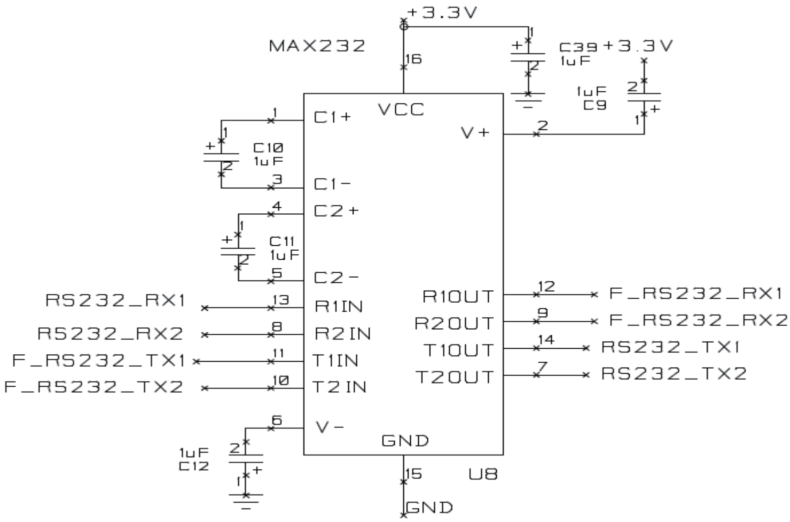


Figure: 6.11 RS 232

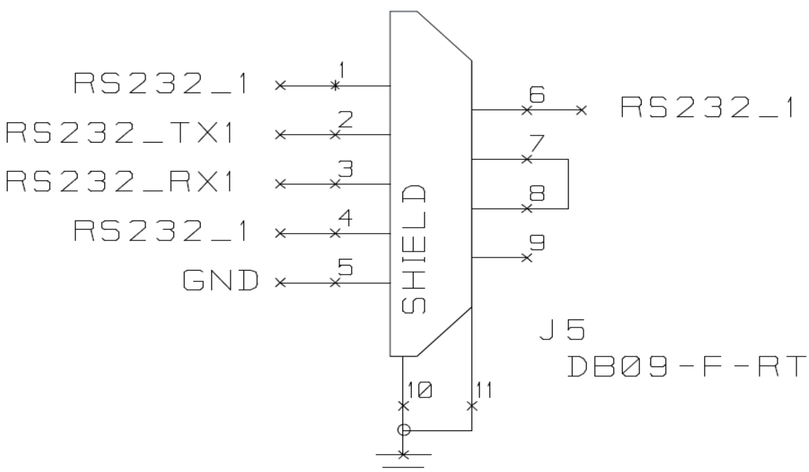


Figure: 6.12 RS 232 CONN

RS232 has been used which is optional. It has been used for testing purpose.

6.8 EHERNET

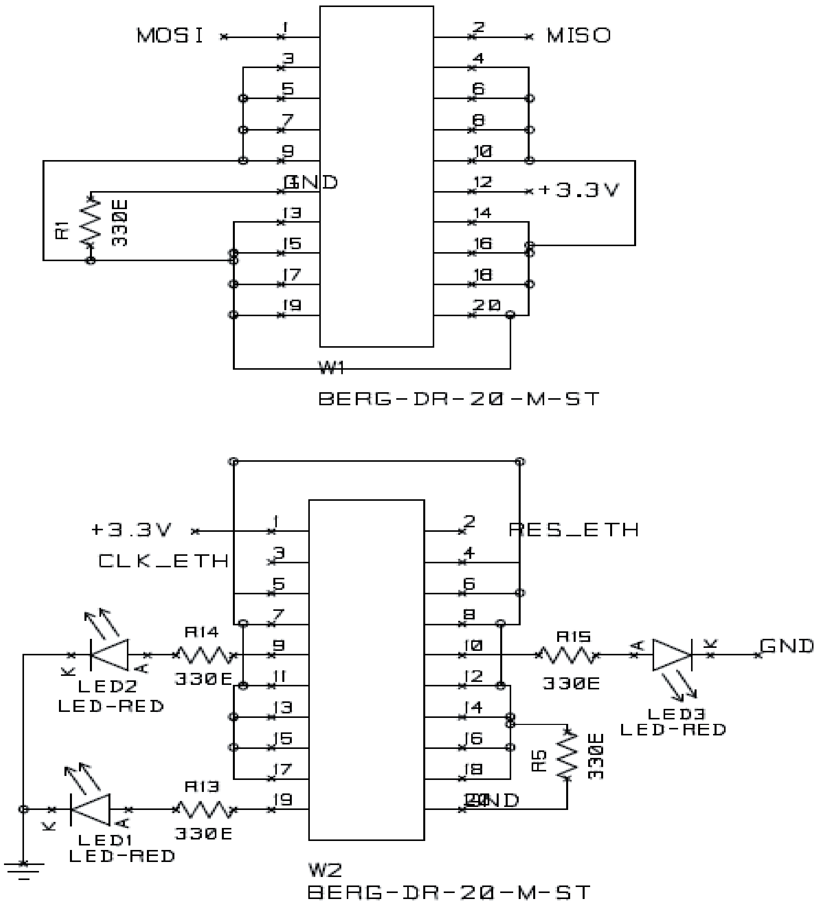


Figure: 6.13 Ethernet

This Ethernet module has been used to mount WIZnet 5100 IC.

6.9 LCD

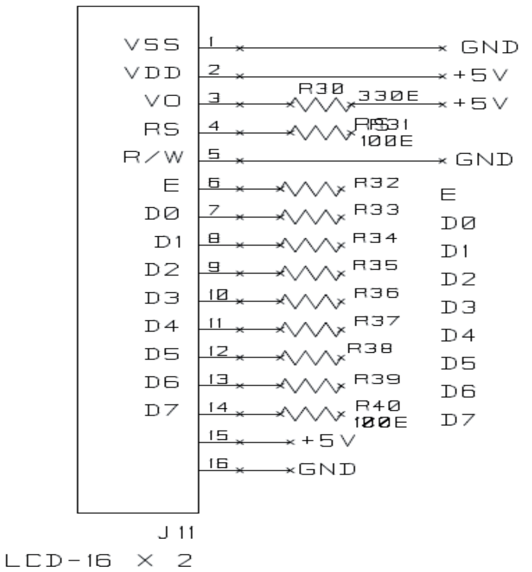


Figure: 6.14 LCD

6.10 JTAG

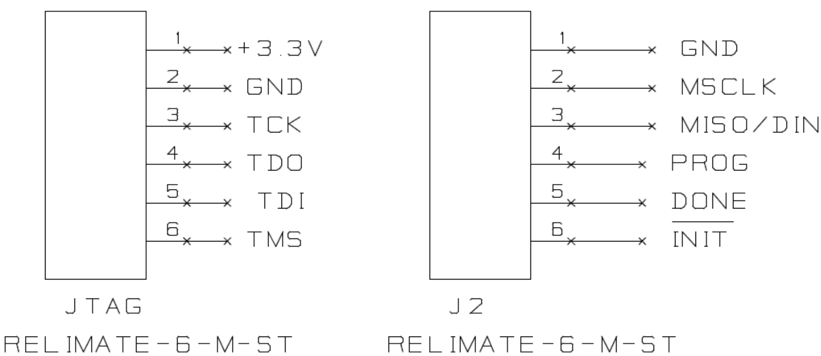


Figure: 6.15 JTAG

JTAG is used for programming into PROM which is located on Controller FPGA board.

Chapter 7

Software Design

Picoblaze

The PicoBlaze microcontroller is a compact, capable, and cost-effective fully embedded 8-bit RISC microcontroller core optimized for the Xilinx FPGA families. In typical implementations, a single FPGA block RAM stores up to 1024 program instructions, which are automatically loaded during FPGA configuration. Even with such resource efficiency, the PicoBlaze microcontroller performs a respectable 44 to 100 million instructions per second (MIPS) depending on the target FPGA family and speed grade.

The PicoBlaze microcontroller core is totally embedded within the target FPGA and requires no external resources. The PicoBlaze microcontroller is extremely flexible. The basic functionality is easily extended and enhanced by connecting additional FPGA logic to the microcontroller's input and output ports. The PicoBlaze microcontroller provides abundant, flexible I/O at much lower cost than off-the-shelf controllers. Similarly, the PicoBlaze peripheral set can be customized to meet the specific features, function, and cost requirements of the target application. Because the PicoBlaze microcontroller is delivered as synthesizable VHDL source code, the core is future-proof and can be migrated to future FPGA architectures, effectively eliminating product obsolescence fears. Being integrated within the FPGA, the PicoBlaze microcontroller reduces board space, design cost, and inventory.

7.1 PicoBlaze Microcontroller Features

The PicoBlaze microcontroller supports the following features:

- 16 byte-wide general-purpose data registers
- 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration
- Byte-wide Arithmetic Logic Unit (ALU) with CARRY and ZERO indicator flags
- 64-byte internal scratchpad RAM
- 256 input and 256 output ports for easy expansion and enhancement

- Automatic 31-location CALL/RETURN stack
- Predictable performance, always two clock cycles per instruction, up to 200 MHz or 100 MIPS in a Virtex-II Pro FPGA
- Fast interrupt response; worst-case 5 clock cycles
- Optimized for Xilinx Spartan-3 architecture—just 96 slices and 0.5 to 1 block RAM
- Support in Spartan-6, and Virtex-6 FPGA architectures
- Assembler, instruction-set simulator support

7.2 Why PicoBlaze Microcontroller?

There are literally dozens of 8-bit microcontroller architectures and instruction sets. Modern FPGAs can efficiently implement practically any 8-bit microcontroller, and available FPGA soft cores support popular instruction sets such as the PIC, 8051, AVR, 6502, 8080, and Z80 microcontrollers. Why use the PicoBlaze microcontroller instead of a more popular instruction set? The PicoBlaze microcontroller is specifically designed and optimized for the Spartan-3 family and with support for Spartan-6 and Virtex-6 FPGA architectures. Its compact yet capable architecture consumes considerably less FPGA resources than comparable 8-bit microcontroller architectures within an FPGA. Furthermore, the PicoBlaze microcontroller is provided as a free, source-level VHDL file with royalty-free re-use within Xilinx FPGAs. Some standalone microcontroller variants have a notorious reputation for becoming obsolete. Because it is delivered as VHDL source, the PicoBlaze microcontroller is immune to product obsolescence as the microcontroller can be retargeted to future generations of Xilinx FPGAs, exploiting future cost reductions and feature enhancements. Furthermore, the PicoBlaze microcontroller is expandable and extendable.

Before the advent of the PicoBlaze and MicroBlaze embedded processors, the microcontroller resided externally to the FPGA, limiting the connectivity to other FPGA functions and restricting overall interface performance. By contrast, the PicoBlaze microcontroller is fully embedded in the FPGA with flexible, extensive on-chip connectivity to other FPGA resources. Signals remain within the FPGA, improving overall performance. The PicoBlaze microcontroller reduces system cost because it is a single-chip solution, integrated within the FPGA and sometimes only occupying leftover FPGA resources. The PicoBlaze microcontroller is resource efficient. Consequently, complex applications are sometimes best portioned across multiple PicoBlaze microcontrollers with each controller implementing a particular function, e.g, keyboard and display control, or system management.

7.3 Why Use a Microcontroller within an FPGA?

Microcontrollers and FPGAs both successfully implement practically any digital logic function. However, each has unique advantages in cost, performance, and ease of use. Microcontrollers are well suited to control applications, especially with widely changing requirements. The FPGA resources required to implement the microcontroller are relatively constant. The same FPGA logic is re-used by the various microcontroller instructions, conserving resources. The program memory requirements grow with increasing complexity. Programming control sequences or state machines in assembly code is often easier than creating similar structures in FPGA logic.

Microcontrollers are typically limited by performance. Each instruction executes sequentially. As the application increases in complexity, the number of instructions required to implement the application grows and system performance decreases accordingly. By contrast, performance in an FPGA is more flexible. For example, an algorithm can be implemented sequentially or completely in parallel, depending on the performance requirements. A completely parallel implementation is faster but consumes more FPGA resources. A microcontroller embedded within the FPGA provides the best of both worlds. The microcontroller implements non-timing crucial complex control functions while timing-critical or data path functions are best implemented using FPGA logic. For example, a microcontroller cannot respond to events much faster than a few microseconds. The FPGA logic can respond to multiple, simultaneous events in just a few to tens of nanoseconds. Conversely, a microcontroller is cost-effective and simple for performing format or protocol conversions.

	PicoBlaze Microcontroller	FPGA Logic
Strengths	• Easy to program, excellent for control and state machine applications • Resource requirements remain constant with increasing complexity • Re-uses logic resources, excellent for lower-performance functions	• Significantly higher performance • Excellent at parallel operations • Sequential vs. parallel implementation trade-offs optimize performance or cost • Fast response to multiple, simultaneous inputs
Weaknesses	• Executes sequentially • Performance degrades with increasing complexity • Program memory requirements increase with increasing complexity • Slower response to simultaneous inputs	• Control and state machine applications more difficult to program • Logic resources grow with increasing complexity

Table: 7.1 PicoBlaze Vs FPGA

7.4 PicoBlaze functional blocks

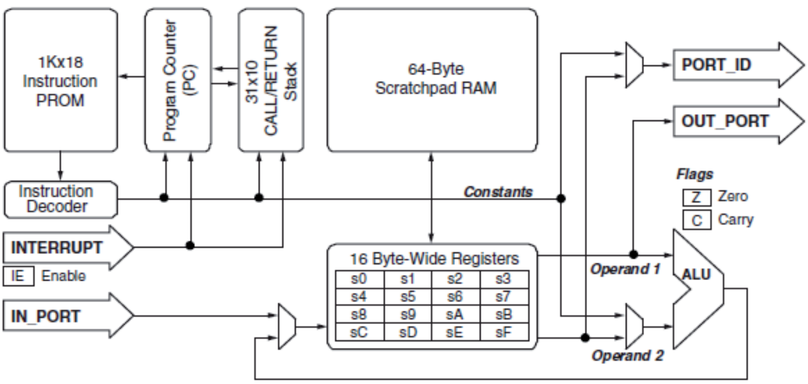


Figure: 7.1 PicoBlaze Embedded Microcontroller Block Diagram

Features:

General-Purpose Registers

The PicoBlaze microcontroller includes 16 byte-wide general-purpose registers, designated as registers s0 through sF. For better program clarity, registers can be renamed using an assembler directive. All register operations are completely interchangeable; no registers are reserved for special tasks or have priority over any other register. There is no dedicated accumulator; each result is computed in a specified register.

1,024-Instruction Program Store

The PicoBlaze microcontroller executes up to 1,024 instructions from memory within the FPGA, typically from a single block RAM. Each PicoBlaze instruction is 18 bits wide. The instructions are compiled within the FPGA design and automatically loaded during the FPGA configuration process. Other memory organizations are possible to accommodate more PicoBlaze controllers within a single FPGA or to enable interactive code updates without recompiling the FPGA design.

Arithmetic Logic Unit (ALU)

The byte-wide Arithmetic Logic Unit (ALU) performs all microcontroller calculations, including:

- Basic arithmetic operations such as addition and subtraction
- Bitwise logic operations such as AND, OR, and XOR
- Arithmetic compare and bitwise test operations
- Comprehensive shift and rotate operations

All operations are performed using an operand provided by any specified register (sX). The result is returned to the same specified register (sX). If an instruction requires a second operand, then the second operand is either a second register (sY) or an 8-bit immediate constant (kk).

Flags

ALU operations affect the ZERO and CARRY flags. The ZERO flag indicates when the result of the last operation resulted in zero. The CARRY flag indicates various conditions, depending on the last instruction executed.

The INTERRUPT_ENABLE flag enables the INTERRUPT input.

64-Byte Scratchpad RAM

The PicoBlaze microcontroller provides an internal general-purpose 64-byte scratchpad RAM, directly or indirectly addressable from the register file using the STORE and FETCH instructions. The STORE instruction writes the contents of any of the 16 registers to any of the 64 RAM locations. The complementary FETCH instruction reads any of the 64 memory locations into any of the 16 registers. This allows a much greater number of variables to be held within the boundary of the processor and tends to reserve all of the I/O space for real inputs and output signals. The six-bit scratchpad RAM address is specified either directly (ss) with an immediate constant, or indirectly using the contents of any of the 16 registers (sY). Only the lower six bits of the address are used; the address should not exceed the 00 - 3F range of the available memory.

Input/Output

The Input/Output ports extend the PicoBlaze microcontroller 's capabilities and allow the microcontroller to connect to a custom peripheral set or to other FPGA logic. The PicoBlaze microcontroller supports up to 256 input ports and 256 output ports or a combination of input/output ports. The PORT_ID output provides the port address. During an INPUT operation, the PicoBlaze microcontroller reads data from the IN_PORT port to a specified register, sX. During an OUTPUT operation, the PicoBlaze microcontroller writes the contents of a specified register, sX, to the OUT_PORT port.

Program Counter (PC)

The Program Counter (PC) points to the next instruction to be executed. By default, the PC automatically increments to the next instruction location when executing an instruction. Only the JUMP, CALL, RETURN, and RETURNI instructions and the Interrupt and Reset Events modify the default behavior. The PC cannot be directly modified by the application code; computed jump instructions are not supported.

The 10-bit PC supports a maximum code space of 1,024 instructions (000 to 3FF hex). If the PC reaches the top of the memory at 3FF hex, it rolls over to location 000.

Program Flow Control

The default execution sequence of the program can be modified using conditional and non-conditional program flow control instructions.

The JUMP instructions specify an absolute address anywhere in the 1,024-instruction program space.

CALL and RETURN instructions provide subroutine facilities for commonly used sections of code. A CALL instruction specifies the absolute start address of a subroutine, while the return address is automatically preserved on the CALL/RETURN stack.

If the interrupt input is enabled, an Interrupt Event also preserves the address of the preempted instruction on the CALL/RETURN stack while the PC is loaded with the interrupt vector, 3FF hex. Use the RETURNI instruction instead of the RETURN instruction to return from the interrupt service routine (ISR).

CALL/RETURN Stack

The CALL/RETURN hardware stack stores up to 31 instruction addresses, enabling nested CALL sequences up to 31 levels deep. Since the stack is also used during an interrupt operation, at least one of these levels should be reserved when interrupts are enabled.

The stack is implemented as a separate cyclic buffer. When the stack is full, it overwrites the oldest value. Consequently, there are no instructions to control the stack or the stack pointer. No program memory is required for the stack.

Interrupts

The PicoBlaze microcontroller has an optional INTERRUPT input, allowing the PicoBlaze microcontroller to handle asynchronous external events. In this context, “asynchronous” relates to interrupts occurring at any time during an instruction cycle. However, recommended design practice is to synchronize all inputs to the PicoBlaze controller using the clock input. The PicoBlaze microcontroller responds to interrupts quickly in just five clock cycles.

Reset

The PicoBlaze microcontroller is automatically reset immediately after the FPGA configuration process completes. After configuration, the RESET input forces the processor into the initial state. The PC is reset to address 0, the flags are cleared, interrupts are disabled, and the CALL/RETURN stack is reset. The data registers and scratchpad RAM are not affected by Reset.

7.5 Source Code

Source Code Platform:

Pico-blaze assembly language will be used on the controller board. Pico-blaze assembler will be used to obtain the respective .vhdl file.

Source code purpose:

The following source code is controlling the communication between the Ethernet module and controller board FPGA and also communication between target FPGA and source FPGA board.

Following is the source code for initialization of W5100 IC

```
LED_VERIFY        DSOUT    5
MISO              DSIN     4
MOSI              DSOUT    9             ; ---------SPI
CS                DSOUT    8             ; ---------SPI
CLK               DSOUT    7             ; ---------SPI
eth_reset         DSOUT    10
CCLK              DSOUT    0
D_IN              DSOUT    1
prog              DSOUT    2
init_b            DSIN     6
done              DSIN     7
RX_DATA           DSIN     5
SW                DSIN     8
TX_DATA           DSOUT    3
TX_ENABLE         DSOUT    4
; ----------------------------- REGISTERS ------------------------- ;
TEMP1             EQU      s0
TEMP2             EQU      s1
TEMP3             EQU      s2
REG               EQU      s3
DATA              EQU      s4
CNTR1             EQU      s5
CNTR2             EQU      s6
SPI_REG           EQU      s7
ADDR1             EQU      s8            ; LSB ADDRESS
ADDR2             EQU      sD
ADDR3             EQU      sA
TEMP              EQU      sB
 ; ----------------------- CONSTANS - INSTRUCTIONS ------------------;
; --------------------------- RAM LOCATIONS ----------------------;

save_s2           EQU      58
```

```
save_s3          EQU     57
save_s4          EQU     56
TX_FLAG          EQU     51
TEMP_FLAG        EQU     40
UR_DATA_RAM      EQU     0
A_1              EQU     $41
B_1              EQU     $42
C_1              EQU     $43
D_1              EQU     $44
E_1              EQU     $45
F_1              EQU     $46
G_1              EQU     $47
H_1              EQU     $48
I_1              EQU     $49
J_1              EQU     $4A
K_1              EQU     $4B
L_1              EQU     $4C
M_1              EQU     $4D
N_1              EQU     $4E
O_1              EQU     $4F
P_1              EQU     $50
Q_1              EQU     $51
R_1              EQU     $52
S_1              EQU     $53
T_1              EQU     $54
U_1              EQU     $55
V_1              EQU     $56
W_1              EQU     $57
X_1              EQU     $58
Y_1              EQU     $59
Z_1              EQU     $5A
BLK_1            EQU     $20
CALL     PROG_B
IN       REG, SW
TEST     REG, 1
JUMP     Z, ABC
JUMP     asd
ABC:
EINT
STOP:            JUMP     STOP
asd:
; load data,$FF
; out data,LED_VERIFY
;
              LOAD     data, $00
              OUT      data, eth_reset
              CALL     DELAY_SMALL
```

```
            LOAD    data, $FF
            OUT     data, eth_reset
sssst:
            LOAD    ADDR2, 0
            LOAD    ADDR1, 0
            LOAD    DATA, 1          ; 144
            CALL    spi_write
; ;;;;;;;;;;;;;;;;;
            LOAD    ADDR2, 1      ;;;;;;Gateway IP Address
            LOAD    ADDR1, 0
            LOAD    DATA, 192
            CALL    spi_write
            LOAD    ADDR2, 2
            LOAD    ADDR1, 0
            LOAD    DATA, 168
            CALL    spi_write
            LOAD    ADDR2, 3
            LOAD    ADDR1, 0
            LOAD    DATA, 0          ; 1
            CALL    spi_write
            LOAD    ADDR2, 4
            LOAD    ADDR1, 0
            LOAD    DATA, 1
            CALL    spi_write; ;;;;;;;;;;;;;;;;;;;;;
            LOAD    ADDR2, 5         ;;;Subnet Mask
            LOAD    ADDR1, 0
            LOAD    DATA, 255
            CALL    spi_write
            LOAD    ADDR2, 6
            LOAD    ADDR1, 0
            LOAD    DATA, 255
            CALL    spi_write
            LOAD    ADDR2, 7
            LOAD    ADDR1, 0
            LOAD    DATA, 255
            CALL    spi_write
            LOAD    ADDR2, 8
            LOAD    ADDR1, 0
            LOAD    DATA, 0
            CALL    spi_write; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            LOAD    ADDR2, 9         ;;;;;;Source Hardware Address
            LOAD    ADDR1, 0
            LOAD    DATA, 0
            CALL    spi_write
            LOAD    ADDR2, 10
            LOAD    ADDR1, 0
            LOAD    DATA, 8
```

```
CALL    spi_write
LOAD    ADDR2, 11
LOAD    ADDR1, 0
LOAD    DATA, $DC
CALL    spi_write
LOAD    ADDR2, 12
LOAD    ADDR1, 0
LOAD    DATA, 1
CALL    spi_write
LOAD    ADDR2, 13
LOAD    ADDR1, 0
LOAD    DATA, 2
CALL    spi_write
LOAD    ADDR2, 14
LOAD    ADDR1, 0
LOAD    DATA, 3
CALL    spi_write ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
LOAD    ADDR2, $0F     ;;;;;;;Source IP Address
LOAD    ADDR1, 0
LOAD    DATA, 192
CALL    spi_write
LOAD    ADDR2, $10
LOAD    ADDR1, 0
LOAD    DATA, 168
CALL    spi_write
LOAD    ADDR2, $11
LOAD    ADDR1, 0
LOAD    DATA, 0            ; 1
CALL    spi_write
LOAD    ADDR2, $12
LOAD    ADDR1, 0
LOAD    DATA, 9
CALL    spi_write ;;;;;;;;;;;;;;;;;;;;;;
LOAD    ADDR2, $01          ; open
LOAD    ADDR1, 4
LOAD    DATA, $01
CALL    spi_write;
LOAD    ADDR2, $1A          ; rx_mem size
LOAD    ADDR1, 0
LOAD    DATA, $06          ; 4kb
CALL    spi_write
LOAD    ADDR2, $1B          ; tx_mem size
LOAD    ADDR1, 0
LOAD    DATA, $55          ; 06
CALL    spi_write
LOAD    ADDR2, $00          ; port 0 is tcp
LOAD    ADDR1, 4
```

```
        LOAD    DATA, 1
        CALL    spi_write
        LOAD    ADDR2, $01      ; LISTEN
        LOAD    ADDR1, 4
        LOAD    DATA, 2
        CALL    spi_write
        LOAD    ADDR2, $04      ; Socket Source Port
        LOAD    ADDR1, 4
        LOAD    DATA, $13
        CALL    spi_write
        LOAD    ADDR2, $05      ; Socket Source Port
        LOAD    ADDR1, 4
        LOAD    DATA, $88
        CALL    spi_write
        LOAD    ADDR2, $12
        LOAD    ADDR1, $04
        LOAD    DATA, $05
        CALL    spi_write
        LOAD    ADDR2, $13
        LOAD    ADDR1, $04
        LOAD    DATA, $B4
        CALL    spi_write
        LOAD    ADDR2, $20
        LOAD    ADDR1, $04
        LOAD    DATA, $08
        CALL    spi_write
        LOAD    ADDR2, $21
        LOAD    ADDR1, $04
        LOAD    DATA, $00
        CALL    spi_write
        LOAD    ADDR2, $26
        LOAD    ADDR1, $04
        LOAD    DATA, $08
        CALL    spi_write
        LOAD    ADDR2, $27
        LOAD    ADDR1, $04
        LOAD    DATA, $00
        CALL    spi_write
        LOAD    ADDR2, $16      ; LISTEN
        LOAD    ADDR1, $00
        LOAD    DATA, $01
        CALL    spi_write
        LOAD    reg, 0
        STORE   reg, 7
data_rcv_int:
        LOAD    ADDR2, $02
        LOAD    ADDR1, $04
```

```
; load DATA,$01
            CALL     spi_read
            OUT      data, LED_VERIFY
            TEST     data, 1
            JUMP     Z, data_rcv_int
; CALL      CONNECTED
data_rcv_intS:
            LOAD     ADDR2, $02
            LOAD     ADDR1, $04
; load DATA,$01
            CALL     spi_read
            OUT      data, LED_VERIFY
            TEST     data, 4
            JUMP     Z, data_rcv_intS
; CALL      big_DELAY
; CALL      big_DELAY
            LOAD     ADDR2, $15          ; LISTEN
            LOAD     ADDR1, $00
            CALL     spi_read
            OUT      data, LED_VERIFY
            TEST     data, 1
            JUMP     Z, data_rcv_intS
; CALL      big_DELAY
; CALL      big_DELAY
; CALL      big_DELAY
; CALL      big_DELAY
; CALL      big_DELAY
            LOAD     ADDR2, $26          ; no of byte rcvd
            LOAD     ADDR1, $04          ; Sn_RX_RSR
            CALL     spi_read
            STORE    data, 62
            LOAD     ADDR2, $27          ; no of byte rcvd
            LOAD     ADDR1, $04          ; Sn_RX_RSR
            CALL     spi_read
            STORE    data, 63
; call data_rcv_intS12
sdf:        LOAD     ADDR2, $26          ; no of byte rcvd
            LOAD     ADDR1, $04          ; Sn_RX_RSR
            CALL     spi_read
            COMP     data, $10
            JUMP     NZ, sdf
            STORE    data, 62
; STORE     data, TX_FLAG
; CALL      DATA_232
            LOAD     ADDR2, $27          ; no of byte rcvd
            LOAD     ADDR1, $04          ; Sn_RX_RSR
            CALL     spi_read
```

```
            STORE    data, 63
; STORE   data, TX_FLAG
; CALL    DATA_232
            LOAD     ADDR2, $28        ; Sn_RX_RD
            LOAD     ADDR1, $04
            CALL     spi_read
            STORE    data, 30
; STORE   data, TX_FLAG
; CALL    DATA_232
            LOAD     ADDR2, $29        ; Sn_RX_RD
            LOAD     ADDR1, $04
            CALL     spi_read
; STORE   data, TX_FLAG
; CALL    DATA_232
            STORE    data, 31
            FETCH    temp1, 30
            FETCH    data, 21        ; 62
            ADD      temp1, data
            STORE    temp1, 30
            FETCH    temp1, 31
            FETCH    data, 22        ; 63
            ADD      temp1, data
            STORE    temp1, 31
; STORE   data, TX_FLAG
; CALL    DATA_232
            LOAD     ADDR2, $02
            LOAD     ADDR1, $04
            LOAD     data, $FF
            CALL     spi_write
            FETCH    data, 30
            AND      data, $0F       ; gSn_RX_MASK
            STORE    data, 2         ; get_offset
            FETCH    data, 31
            AND      data, $FF       ; gSn_RX_MASK
            STORE    data, 3         ; get_offset
            FETCH    temp1, 2
            FETCH    temp2, 3
            ADD      temp2, 0
            ADD      temp1, $60
            STORE    temp1, 2          ; get_start_address
            STORE    temp2, 3          ; get_start_address
            FETCH    ADDR2, 3
            FETCH    ADDR1, 2
; ADD   ADDR1, 2
            FETCH    temp1, 62
            FETCH    temp2, 63
            STORE    temp1, 21
```

```
                STORE     temp2, 22
tp:
                CALL      spi_read
; STORE     data, TX_FLAG
; CALL      DATA_232
                LOAD      SPI_REG, DATA
                CALL      CONFIG_DATA
                FETCH     ADDR2, 3
                FETCH     ADDR1, 2
                ADD       addr2, 1
                ADDC      addr1, 0
                STORE     ADDR2, 3
                STORE     ADDR1, 2
;  fetch temp1,62
; fetch temp2,63
;
;               FETCH     temp1, 62
                FETCH     temp2, 63;
                SUB       temp2, 1
                SUBC      temp1, 0;
                STORE     temp1, 62
                STORE     temp2, 63
                COMP      temp2, 0
                JUMP      Z, tt6
                JUMP      ag88;
tt6:
                COMP      temp1, 0
                JUMP      Z, asd15
;
;
ag88:
                JUMP      tp
asd15:

                LOAD      ADDR2, 0
                LOAD      ADDR1, 0
                LOAD      DATA, 128         ; 144
                CALL      spi_write
                LOAD      ADDR2, $28        ; Sn_RX_RD
                LOAD      ADDR1, $04
                CALL      spi_read
                STORE     data, 30
                LOAD      ADDR2, $29        ; Sn_RX_RD
                LOAD      ADDR1, $04
                CALL      spi_read
                STORE     data, 31
                FETCH     data, 30
```

```
            AND      data, $0F          ; gSn_RX_MASK
            STORE    data, 2            ; get_offset
            FETCH    data, 31
            AND      data, $FF          ; gSn_RX_MASK
            STORE    data, 3            ; get_offset
            LOAD     ADDR2, $26         ; no of byte rcvd
            LOAD     ADDR1, $04         ; Sn_RX_RSR
            CALL     spi_read
            STORE    data, 62
            LOAD     ADDR2, $27         ; no of byte rcvd
            LOAD     ADDR1, $04         ; Sn_RX_RSR
            CALL     spi_read
            STORE    data, 63
;
            JUMP     sssst
askjd:      JUMP     askjd
DATA_232:
            FETCH    TEMP1, TX_FLAG
            OUT      TEMP1, TX_DATA
            LOAD     TEMP1, $FF
            OUT      TEMP1, TX_ENABLE
; CALL     DELAY_SMALL
; CALL     DELAY_SMALL
            CALL     DELAY_SMALL
            CALL     DELAY_SMALL
            CALL     DELAY_SMALL
; CALL     DELAY
            LOAD     TEMP1, $00
            OUT      TEMP1, TX_ENABLE
            CALL     DELAY
; CALL     DELAY_SMALL
            CALL     DELAY_SMALL
            CALL     DELAY_SMALL
            RET
; RET
; --------------------------- WRITE status register ----------------------------
spi_write:     LOAD     temp, 0
            OUT      temp, cs
            LOAD     cntr1, 0
            LOAD     SPI_REG, $F0       ; REN
            CALL     spi_tx
            LOAD     SPI_REG, ADDR1
            CALL     spi_tx
            LOAD     SPI_REG, ADDR2
            CALL     spi_tx
            LOAD     SPI_REG, DATA
            CALL     spi_tx
```

```
            LOAD    temp, 1
            OUT     temp, cs
            RET
spi_read:       LOAD    temp, 0
            OUT     temp, cs
            LOAD    cntr1, 0
            LOAD    SPI_REG, $0F        ; WREN
            CALL    spi_tx
            LOAD    SPI_REG, ADDR1
            CALL    spi_tx
            LOAD    SPI_REG, ADDR2
            CALL    spi_tx
; LOAD     SPI_REG, DATA
; CALL     spi_tx
            CALL    spi_rx
            LOAD    temp, 1
            OUT     temp, cs
            RET
spi_tx:         OUT     SPI_REG, MOSI       ;
            CALL    clk_pulse       ; SET
            SL0     SPI_REG         ;
            ADD     cntr1, 1        ;
            COMP    cntr1, 8        ;
            JUMP    C, spi_tx       ;
            LOAD    cntr1, 0        ;
            RET                     ;
; ---------------------------receive ----------------------------------------

spi_rx:     IN      SPI_REG, MISO       ;
            CALL    clk_pulse
            SL0     SPI_REG
            SLA     data
            ADD     cntr1, 1
            COMP    cntr1, 8
            JUMP    C, spi_rx
            LOAD    cntr1, 0
            RET
; --------------------------------------------------------------------------
clk_pulse:      LOAD    temp, 1         ;;;;;;
            OUT     temp, clk
            LOAD    temp, 0
            OUT     temp, clk
            RET
big_DELAY:      STORE   s2, save_s2
            STORE   s3, save_s3
            LOAD    s2, $FA
            LOAD    s3, 10
```

```
            CALL    DELAY_large
WAIT33:     SUB     s2, 1
            JUMP    NZ, WAIT33
            SUB     s3, 1
            LOAD    s2, $FA
            JUMP    NZ, WAIT33
            FETCH   s2, save_s2
            FETCH   s3, save_s3
            RET
DELAY:          STORE   s2, save_s2
            STORE   s3, save_s3
            LOAD    s2, $FA
            LOAD    s3, 10          ; L7;
WAIT:           SUB     s2, 1
            JUMP    NZ, WAIT
            SUB     s3, 1
            LOAD    s2, $FA
            JUMP    NZ, WAIT
            FETCH   s2, save_s2
            FETCH   s3, save_s3
            RET
DELAY_large:        STORE   s6, save_s4
            LOAD    s6, 7
WAIT2:          CALL    DELAY
            SUB     s6, 1
            JUMP    NZ, WAIT2
            FETCH   s6, save_s4
            RET
DELAY_SMALL:        STORE   s6, save_s2
            LOAD    s6, 200
WAIT3:          SUB     s6, 1
            JUMP    NZ, WAIT3
            FETCH   s6, save_s2
            RET
CONNECTED:
            LOAD    TEMP1, C_1
            STORE   TEMP1, TX_FLAG
            CALL    DATA_232
            LOAD    TEMP1, O_1
            STORE   TEMP1, TX_FLAG
            CALL    DATA_232
            LOAD    TEMP1, N_1
            STORE   TEMP1, TX_FLAG
            CALL    DATA_232
            LOAD    TEMP1, N_1
            STORE   TEMP1, TX_FLAG
            CALL    DATA_232
```

```
            LOAD     TEMP1, E_1
            STORE    TEMP1, TX_FLAG
            CALL     DATA_232
            LOAD     TEMP1, C_1
            STORE    TEMP1, TX_FLAG
            CALL     DATA_232
            LOAD     TEMP1, T_1
            STORE    TEMP1, TX_FLAG
            CALL     DATA_232
            LOAD     TEMP1, E_1
            STORE    TEMP1, TX_FLAG
            CALL     DATA_232
            LOAD     TEMP1, D_1
            STORE    TEMP1, TX_FLAG
            CALL     DATA_232
            RET
PROG_B:
            LOAD     temp1, $00
            OUT      temp1, prog
            LOAD     temp1, $FF
            OUT      temp1, prog
            RET
data_rcv_intS12:
            LOAD     ADDR2, $02
            LOAD     ADDR1, $04
            CALL     spi_read
            OUT      data, LED_VERIFY
            TEST     data, 4
            JUMP     NZ, data_rcv_intS12
            RET
CCLK_pulse:       LOAD     temp, 1          ;;;;;;
            OUT      temp, CCLK
            LOAD     temp, 0
            OUT      temp, CCLK
            RET
CONFIG_DATA:      OUT      SPI_REG, D_IN       ;
            CALL     CCLK_pulse         ; SET
            SL0      SPI_REG          ;
            ADD      cntr1, 1         ;
            COMP     cntr1, 8         ;
            JUMP     C, CONFIG_DATA     ;
            LOAD     cntr1, 0         ;
            RET                       ;
CHK_STR:
            COMP     SPI_REG, $2A
            JUMP     NZ, ISR
            LOAD     REG, 78
```

```
                STORE    REG, TEMP_FLAG
                CALL     PROG_B
                JUMP     ISR
ISR1:
                IN       SPI_REG, RX_DATA
                OUT      SPI_REG, LED_VERIFY
;               FETCH    REG, TEMP_FLAG
;               COMP     REG, 78
;               JUMP     NZ, CHK_STR
                CALL     CONFIG_DATA
;               IN       REG, done
;               TEST     REG, 1
;               JUMP     NZ, ISR
;               LOAD     REG, 0
;               STORE    REG, TEMP_FLAG

ISR:                              ; CALL    UR_DATA       ; -------------        ;----
INTRUPT
                RETI     ENABLE
                ORG      $3FF
                JUMP     ISR1
```

FLOW CHART:

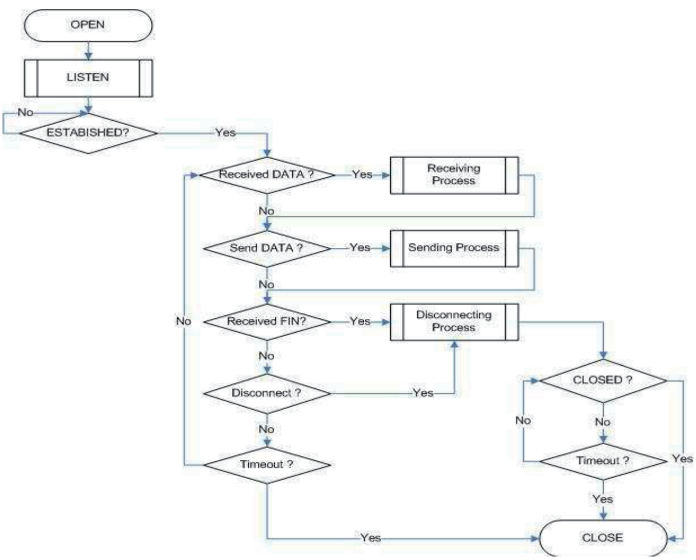


Figure: 7.2 Flow Chart of W5100 in Server mode

Chapter 8

Experimentation

TESTING

After discussing the design details above the board is tested to verify the proper functioning of the system. Following are the results :

8.1 Initialization of W5100

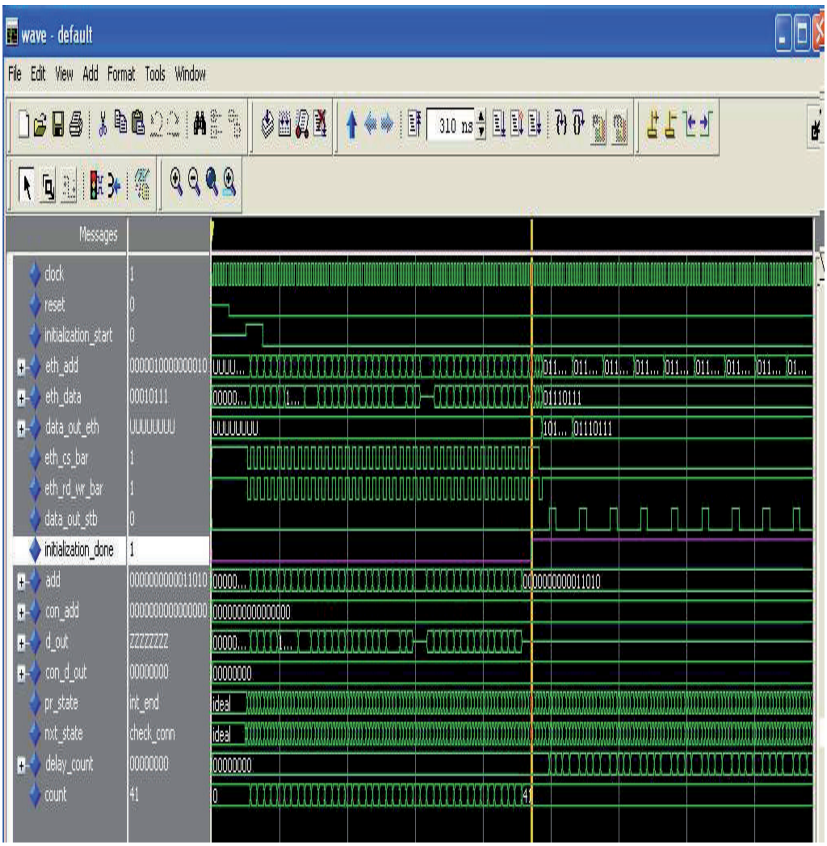


Figure: 8.1 Initialization of W5100

8.2 FPGA Configuration

I)

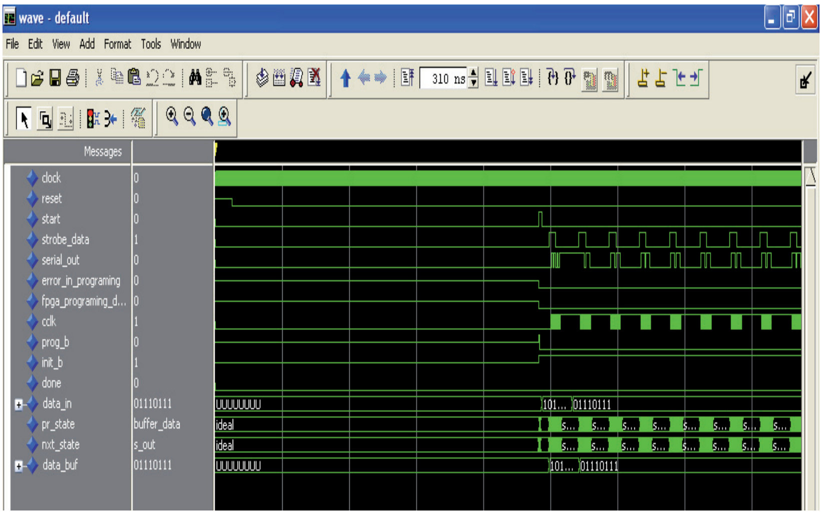


Figure: 8.2 FPGA configuration I

II)

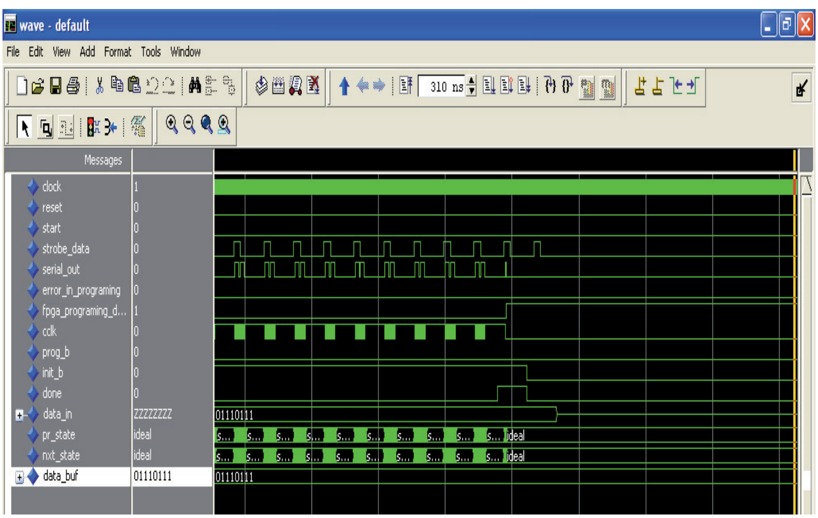
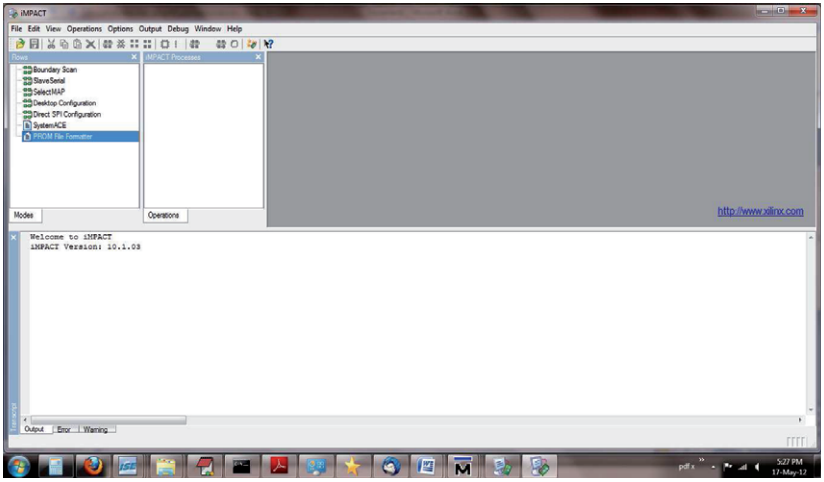


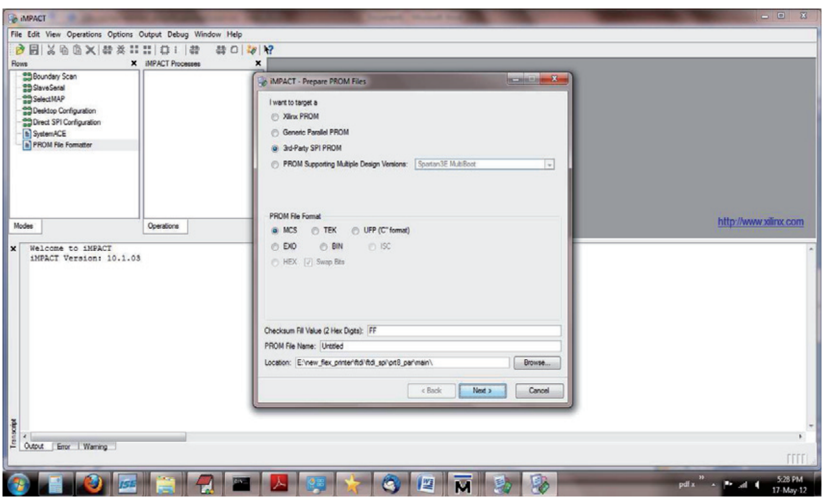
Figure: 8.3 FPGA configuration II

8.3 Procedure

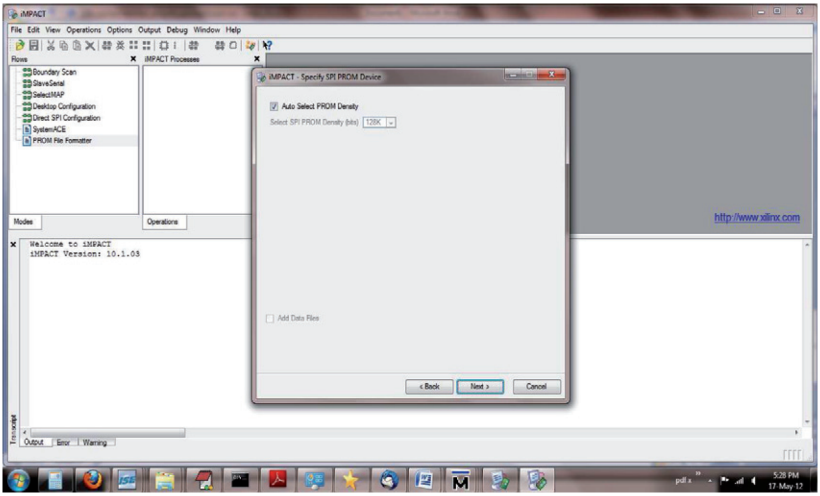
Step 1



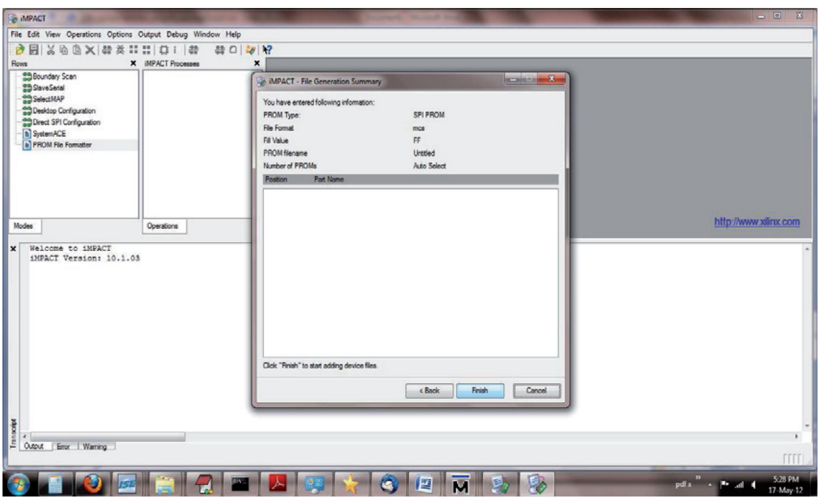
Step 2



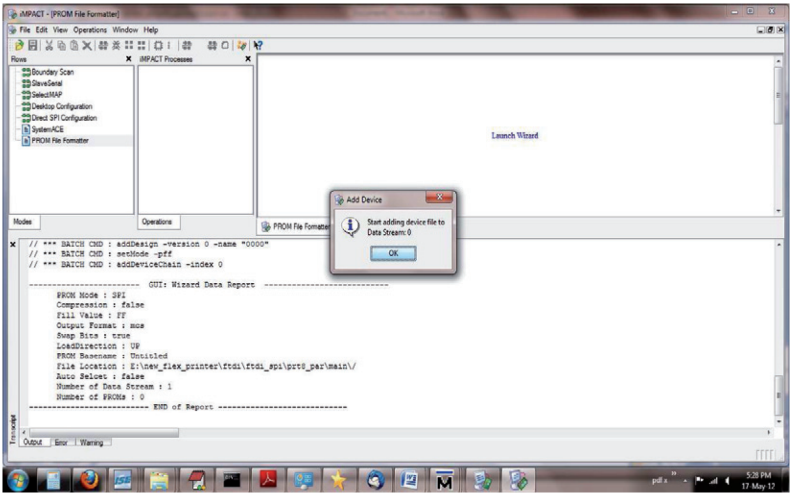
Step 3


iMPACT
File Edit View Operations Options Output Debug Window Help
Flows
iMPACT Processes
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
iMPACT - Specify SPI PROM Device
Auto Select PROM Density
Select SPI PROM Density (bits) 128K
Add Data Files
< Back
Next >
Cancel
http://www.xilinx.com
Modes
Operations
Welcome to iMPACT
iMPACT Version: 10.1.03
Output
Error
Warning
5:28 PM
17-May-12

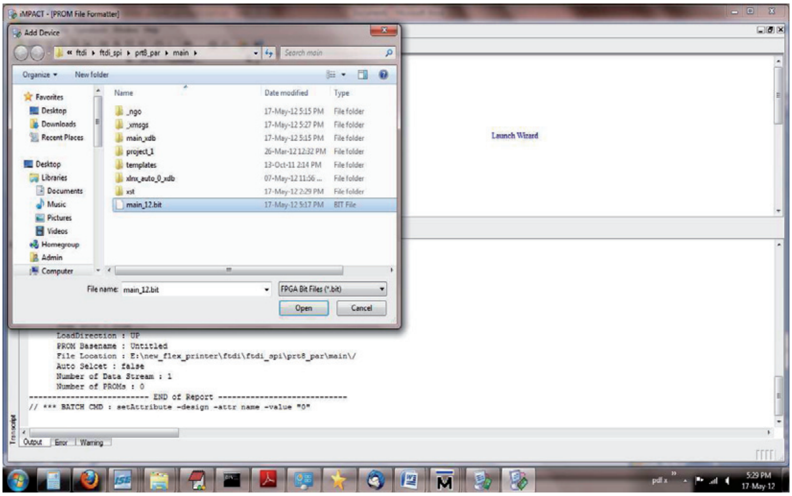
Step 4


iMPACT
File Edit View Operations Options Output Debug Window Help
Flows
iMPACT Processes
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
iMPACT - File Generation Summary
You have entered following information:
PROM Type: SPI PROM
File Format mcs
Fill Value FF
PROM filename Untitled
Number of PROMs Auto Select
Position
Part Name
Click "Finish" to start adding device files.
< Back
Finish
Cancel
http://www.xilinx.com
Modes
Operations
Welcome to iMPACT
iMPACT Version: 10.1.03
Output
Error
Warning
5:28 PM
17-May-12

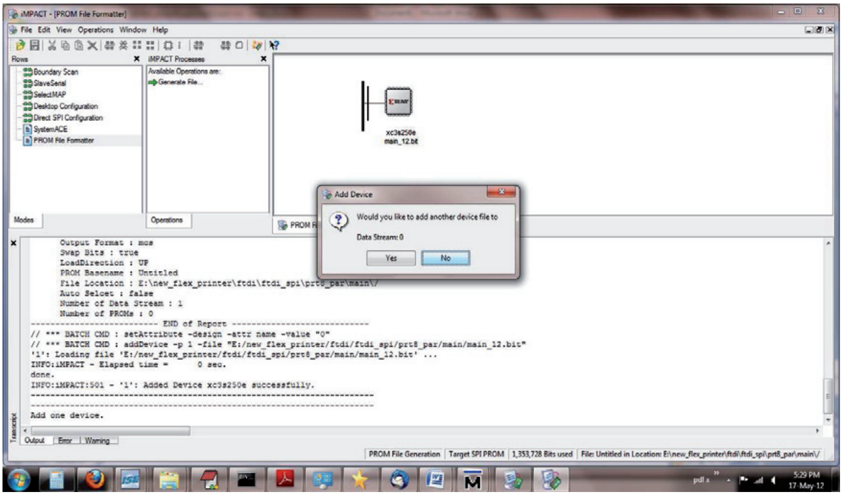
Step 5



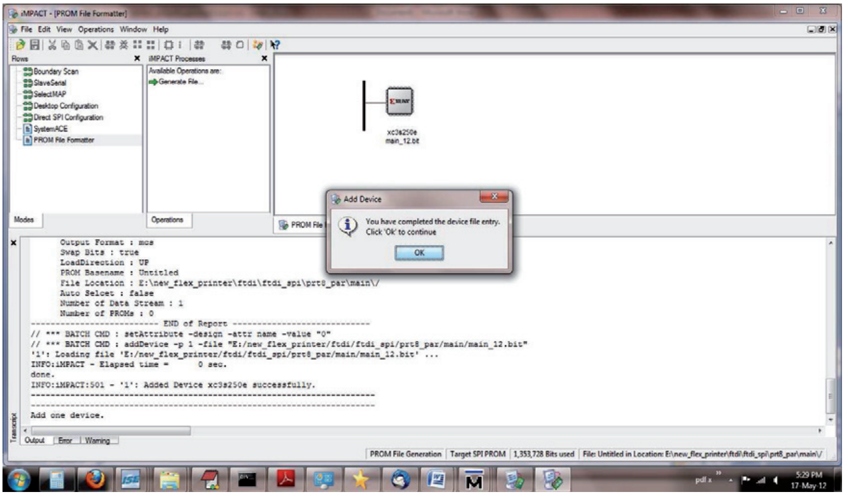
Step 6



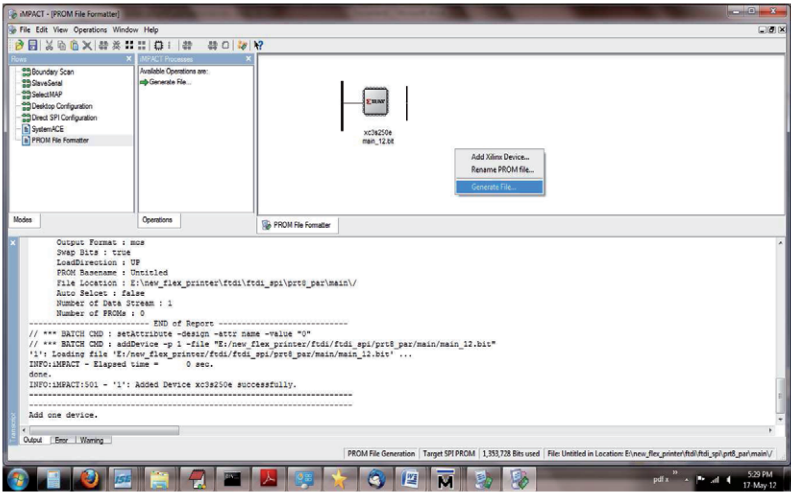
Step 7


iMPACT - [PROM File Formatter]
File Edit View Operations Window Help
Flows
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
iMPACT Processes
Available Operations are:
Generate File...
xc3s250e
main_12.bit
Add Device
Would you like to add another device file to
Data Stream: 0
Yes
No
Output Format : mcs
Swap Bits : true
LoadDirection : UP
PROM Basename : Untitled
File Location : E:\new_flex_printer\ftdi\ftdi_spi\prt8_par\main\/
Auto Select : false
Number of Data Stream : 1
Number of PROMs : 0
END of Report
// *** BATCH CMD : setAttribute -design -attr name -value "0"
// *** BATCH CMD : addDevice -p 1 -file "E:/new_flex_printer/ftdi/ftdi_spi/prt8_par/main/main_12.bit"
'1': Loading file 'E:/new_flex_printer/ftdi/ftdi_spi/prt8_par/main/main_12.bit' ...
INFO:iMPACT - Elapsed time = 0 sec.
done.
INFO:iMPACT:501 - '1': Added Device xc3s250e successfully.
Add one device.
Output
Error
Warning
PROM File Generation
Target SPI PROM
1,353,728 Bits used
File: Untitled in Location: E:\new_flex_printer\ftdi\ftdi_spi\prt8_par\main\/
5:29 PM
17-May-12

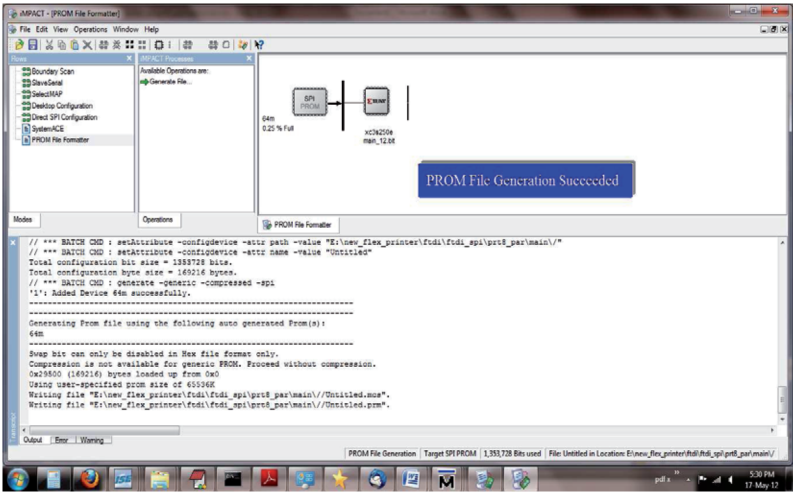
Step 8


iMPACT - [PROM File Formatter]
File Edit View Operations Window Help
Flows
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
iMPACT Processes
Available Operations are:
Generate File...
xc3s250e
main_12.bit
Add Device
You have completed the device file entry.
Click 'Ok' to continue
OK
Output Format : mcs
Swap Bits : true
LoadDirection : UP
PROM Basename : Untitled
File Location : E:\new_flex_printer\ftdi\ftdi_spi\prt8_par\main\/
Auto Select : false
Number of Data Stream : 1
Number of PROMs : 0
END of Report
// *** BATCH CMD : setAttribute -design -attr name -value "0"
// *** BATCH CMD : addDevice -p 1 -file "E:/new_flex_printer/ftdi/ftdi_spi/prt8_par/main/main_12.bit"
'1': Loading file 'E:/new_flex_printer/ftdi/ftdi_spi/prt8_par/main/main_12.bit' ...
INFO:iMPACT - Elapsed time = 0 sec.
done.
INFO:iMPACT:501 - '1': Added Device xc3s250e successfully.
Add one device.
Output
Error
Warning
PROM File Generation
Target SPI PROM
1,353,728 Bits used
File: Untitled in Location: E:\new_flex_printer\ftdi\ftdi_spi\prt8_par\main\/
5:29 PM
17-May-12

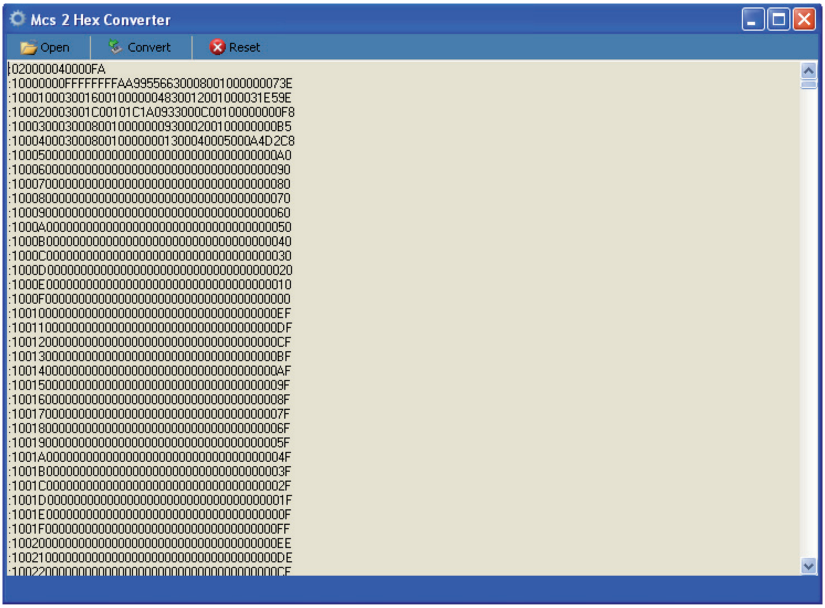
Step 9



Step 10

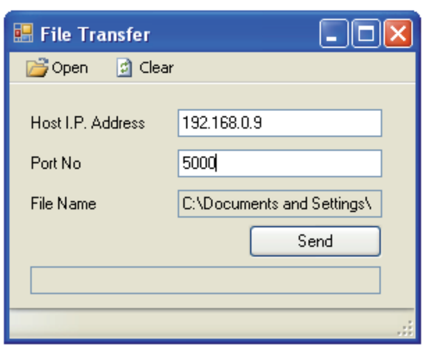


Step 11
Converting **.mcs** file to **.hex** file



Step 12

Finally transferring data by providing IP address of Controller board



Chapter 9

Conclusion

We Can Configure Remotely placed FPGA using *Slave Serial Mode* and *Ethernet IC W5100.*

9.1 Benefit of Work

It provides a more efficient use of utilizing a single FPGA board for hardware simulation. It could also place itself in educational establishments where the prototyping board does not need to be changed from workstation to workstation and also provides an efficient way of managing expensive FPGA resources. Now a day, this type of multi-user FPGA board use has much further development until it becomes a viable multi-userhardware simulation environment.

Such a strategy would bring benefits to FPGA rapid prototyping engineers, engineers changing workstations or on the move, and distributed design teams posted around the globe. It could also place itself in educational establishments where the prototyping board does not need to be changed from workstation to workstation and also provides an efficient way of managing expensive FPGA resources.

9.2 Final Results

Different programs can be run on target FPGA board. Following are the two application oriented programs that has been successfully implemented and verified using this project.

I) Traffic light Controller(TLC)

Following is the source code for the same using VHDL language:

```
library ieee;
use ieee.std_logic_1164.all;


ENTITY traffic_light IS
 PORT(sensor : IN std_logic;
     clock : IN std_logic;
     red_light : OUT std_logic;
     green_light : OUT std_logic;
```

```
      yellow_light : OUT std_logic);
END traffic_light;

ARCHITECTURE simple OF traffic_light IS
  TYPE t_state is (red, green, yellow);
  SIGNAL present_state, next_state : t_state;
BEGIN
  PROCESS(present_state, sensor)
  BEGIN
    CASE present_state IS
      WHEN green =>
        next_state <= yellow;
        red_light <= '0';
        green_light <= '1';
        yellow_light <= '0';
      WHEN red =>
        red_light <= '1';
        green_light <= '0';
        yellow_light <= '0';
        IF (sensor = '1') THEN
          next_state <= green;
        ELSE
          next_state <= red;
        END IF;
      WHEN yellow =>
        red_light <= '0';
        green_light <= '0';
        yellow_light <= '1';
        next_state <= red;
    END CASE;
  END PROCESS;

  PROCESS
  BEGIN
    WAIT UNTIL clock'EVENT and clock = '1';
    present_state <= next_state;
  END PROCESS;
END simple;
```

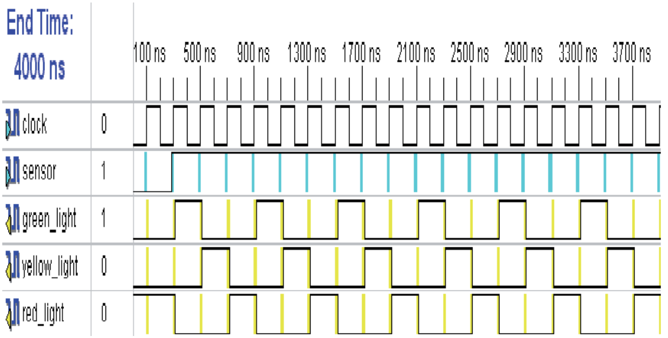


Figure: 9.1 Testbench Waveform for TLC

II) 64 Bit RAM(16*4)

Following is the source code for the same using VHDL language:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY ram IS
    GENERIC
    (
            ADDRESS_WIDTH   : integer := 4;
            DATA_WIDTH      : integer := 8
    );
    PORT
    (
            clock: IN  std_logic;
            data: IN  std_logic_vector(DATA_WIDTH - 1 DOWNTO 0);
            write_address: IN  std_logic_vector(ADDRESS_WIDTH - 1
DOWNTO 0);
            read_address: IN  std_logic_vector(ADDRESS_WIDTH - 1
DOWNTO 0);
            we: IN  std_logic;
            q: OUT std_logic_vector(DATA_WIDTH - 1 DOWNTO 0)
    );
END ram;
ARCHITECTURE rtl OF ram IS
    TYPE RAM IS ARRAY(0 TO (2 ** ADDRESS_WIDTH)-1) OF
std_logic_vector(DATA_WIDTH - 1 DOWNTO 0);
    SIGNAL ram_block : RAM;
BEGIN
    PROCESS (clock)
    BEGIN
```

```
            IF (clock'event AND clock = '1') THEN
                IF (we = '1') THEN
                    ram_block(to_integer(unsigned(write_address))) <= data;
                        q<="00000000";
                END IF;
                q <= ram_block(to_integer(unsigned(read_address)));
            END IF;
        END PROCESS;
    END rtl;
```

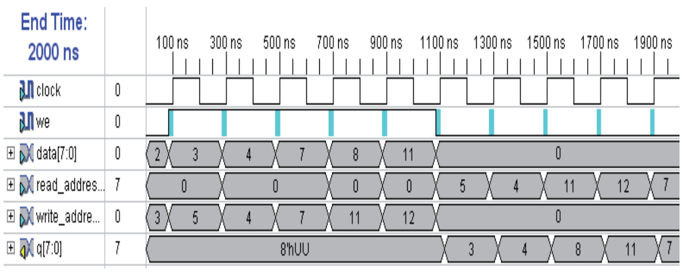


Figure: 9.2 Testbench waveform for RAM

9.3 COMPONENT COSTING

S.No.	*Component Name*	*Cost (Rs)*	*Quantity*
1.	Ethernet Module	4000	1
2.	FPGA	1000	2
3.	PCB	800	2
4.	Casing	400	1
5.	PROM	300	1
6.	Power Supply	250	2
7.	LCD	200	1
8.	MAX 3232	100	1
9.	DB 9 Pin	50	1
10.	LM 317	35	6
11.	Miscellaneous	500	1
		Total :	**Rs. 9860/-**

9.4 FUTURE SCOPE

Future work, include the management of multiple boards and modules on a rack with only one TCP/IP address or Wi-Fi/ Wi-Max. This would allow remote control of a rack in a laboratory. If two or more users are waiting for a board and there are further constant requests for hardware co- simulation, the assignment of the next user is very much random and there is the possibility for a user never to acquire the board and to always be queuing means Deadlock can be there.

To avoiding these problems we can use different available protocols supported by this process. This is currently unacceptable so therefore some sort of queuing stack needs to be implemented. Further milestones include queuing of multiple hardware co-simulations with an option for priorities and also batch processing of models. Early work also continues on the use of FPGA device management, whereby devices on a board populated with many FPGAs can be targeted individually.

References:

I) Books:

[1] "VHDL Primer" by J. Bhaskar,3rd edition, Prentice Hall .

[2] "VHDL Programming by example" by Douglas Perry, Tata McGraw Hill.

[3] "Designing with FPGA and CPLD" by J.Jenkins, Prentice Hall.

II) Papers:

[1] Fallside, Smith," Internet Connected FPGAs", In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, Napa Valley, USA, April 2000.

[2] Brebner, G., Diessel, O., "Chip-Based Reconfigurable Task Management", In Proceedings of the 11th International Conference on Field Programmable Logic 2001, Belfast, Northern Ireland, August 2001.

[3] Jim Burns, Adam Donlin, Jonathan Hogg, Satnam Singh, Mark de, "A Dynamic reconfiguration run time system" in the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, 1997 Proceedings.

[4] R. Lysecky and F. Vahid, "A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning," in Proc. Des. Autom. and Test Eur. Conf., Mar. 2005, pp. 18-23. [13] R. Scrofano, S. Choi, and V. K. Prasanna, "Energy efficiency of FPGAs and programmable processors for matrix multiplication," in Proc. Int. Conf. Field Programmable Technol., Dec. 2002, pp. 422-425.

[5] "SLOPES: Hardware-Software Co synthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs" in IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems, VOL. 26, NO. 3, MARCH 2007

[6] Piyush Kumar Shukla, Dr.S.Silakari, Dr.Sarita.S.Bhadoria, Prof. Anuj Garg," Multi-User FPGA - An Efficient Way of Managing Expensive FPGA Resources Using TCP/IP, Wi-Max/ Wi-Fi in a Secure Network Environment" in Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping

[7] Daniel Denning, James Irvine, Derek Stark, Malachy Devlin "Multi-User FPGA Co-Simulation Over TCP/IP" in Proceedings of the IEEE International Conference on Field-Programmable Custom Computing Machines.

III) WEB Reference:

[1] PicoBlaze 8-bit Embedded Microcontroller User Guide, http://www.xilinx.com /support/documentation/ip_documentation/ug129.pdf

[2] WIZnet datasheet, http://www.wiznet.co.kr/UpLoadFiles/ReferenceFiles/W5100_ Datasheet_v1.2.2.pdf

Publication & Conferences

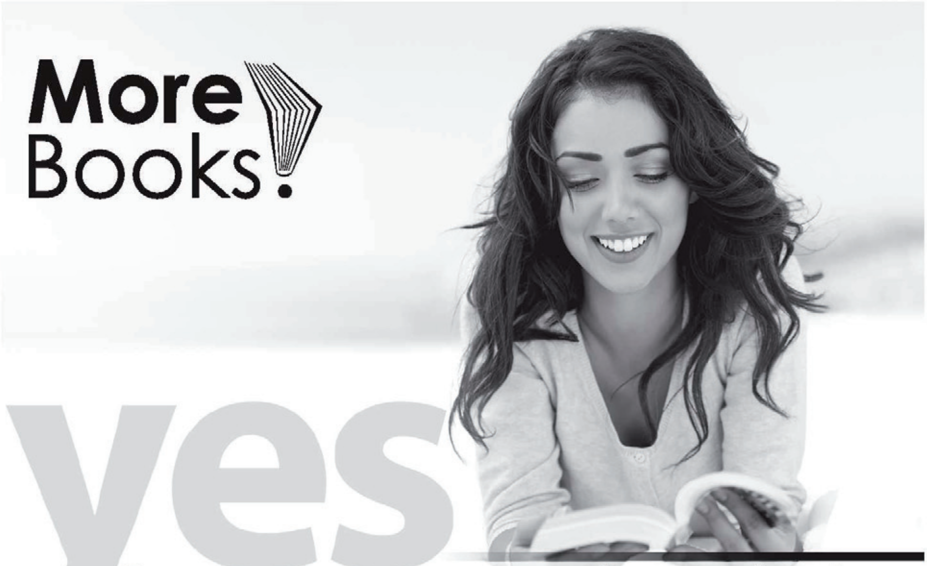
"IMPLEMENTATION OF MULTI-USER FPGA ENVIRONMENT OVER TCP-IP"

1) Above paper has been published in **"International Journal of Engineering and Innovative Technology (ISSN: 2277-3754)"** in Volume 1 Issue 4, April 2012.
   Above paper can be downloaded from following link:

   http://www.ijeit.com/vol%201/Issue%204/IJEIT1412201204_24.pdf

2) Above paper has been presented in **"International Conference on Novel Horizons & Prospects Of Industry Institute Interaction"** at Shri Shankarprasad Agnihotri College Of Engineering,Wardha.

3) Presented paper in **"ePGCON 2012"** at Cummins College Of Engineering,Pune.



I want morebooks!

Buy your books fast and straightforward online - at one of the world's fastest growing online book stores! Environmentally sound due to Print-on-Demand technologies.

Buy your books online at

www.get-morebooks.com

Kaufen Sie Ihre Bücher schnell und unkompliziert online – auf einer der am schnellsten wachsenden Buchhandelsplattformen weltweit!
Dank Print-On-Demand umwelt- und ressourcenschonend produziert.

Bücher schneller online kaufen

www.morebooks.de

OmniScriptum Marketing DEU GmbH
Heinrich-Böcking-Str. 6-8
D - 66121 Saarbrücken
Telefax: +49 681 93 81 567-9

info@omniscriptum.com
www.omniscriptum.com

OMNIScriptum