1-1-2007

Embedded e-maintenance for an FPGA-based reconfigurable system

Dina Goldenberg
Ryerson University

b1819476X

Embedded e-Maintenance for an FPGA-based reconfigurable system

By

Dina Goldenberg

A project
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of

Master of Engineering

in the department of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2007

Author's Declaration:

Borrow List

Ryerson University requires the signatures of all persons using or photocopying this project.

Please sign below, and give address and date.

Name	Signature	Address	Date

Acknowledgement

I would like to express my deep gratitude to my supervisor Professor Lev Kirischian for his support and encouragement during my M.Eng project. I am very grateful for his guidance and valuable advice.

I would also like to thank the Department of Electrical and Computer Engineering and Embedded Reconfigurable Systems Lab (ERSL) for providing facilities and resources necessary for this project. My special thanks to Mr. Valeri Kirischian for his technical assistance during the experimental phase of this project.

Embedded e-Maintenance for an FPGA-based reconfigurable system

Abstract

In recent years with the use of Internet Technologies e-Maintenance systems for remote connectivity, performance monitoring and diagnostics were introduced. As reconfigurable systems based on FPGAs are becoming more and more popular due to their low time-to-market and reprogrammable feature, new possibilities for e-Maintenance are opened allowing remote repair of the system by sending new firmware via Internet for reconfiguration of FPGA. Up until recently programming of FPGA has been a complicated hardware design process. However, as FPGAs were evolving, their reconfiguration time was significantly reduced and partial reconfiguration became available, application programming into FPGA can be simplified as presented in [1]. This method is based on temporal partitioning of FPGA and periodically reloading it with segments of application for different tasks. This allows utilization of smaller and thus much cheaper FPGA and also simplifies the programming. With the help of e-Maintenance the whole system can be dynamically reconfigured. Remote programmer can perform partial reconfiguration, chose the physical location in FPGA for reconfiguration and upgrade different segments.

In this project a research of remote maintenance and reconfigurable systems is conducted and an e-Maintenance system is developed for an FPGA-based platform.

Contents

List of Figures

List of Tables

Chapter One

Introduction

1.1 Motivation

Maintenance of a system is the next important task after manufacturing and production. Since system reliability is crucial for any sold product and customer satisfaction is of a greatest value in the modern commercial world, the system maintenance must be effective and inexpensive. In large complex systems reliability is difficult to achieve since many boards and sub-blocks are involved in system design and integration and therefore probability of malfunctioning and faults is high. When system malfunctioning is detected in the field, the work to repair the system can be simplified if some information about malfunctioning is known. Saving data to log files that later on will be viewed by technicians can shed light on the source of the problem. Self check and self diagnostic subsystems are designed and added to complex systems in order to facilitate maintenance process. Latest Internet technologies have enabled remote connectivity to the system which opens huge opportunities for design of remote e-Maintenance systems to perform remote monitoring, remote access to log files and diagnostic. With new possibility to reconfigure FPGA systems by downloading new firmware some remote repair work became possible.

FPGA-based systems carry big advantages over ASIC systems due to possibility to be reconfigured and adapt to different requirements for different applications. Due their small sizes and high density they are vulnerable to external disturbances, which can cause temporary or permanent damage. In case of SEU (Single Event Upset) the damage might be not of a permanent nature and by simply reloading FPGA the problem can be fixed.

Also Internet connection opens possibilities for remote reconfiguration of the system for new applications. As modern FPGA require significantly less time for reconfiguration and posses the ability of partial reconfiguration constant periodic reconfiguration of FPGA for different tasks becomes possible allowing utilization of smaller FPGA for complex systems. These tasks can be programmed remotely and uploaded to FPGA partitions with the help of e-Maintenance system.

1.2 Project Objective

The objective of this project is to conduct research in the area of e-Maintenance, which includes system maintenance and Internet communication. FPGA-based systems study has to be conducted as well.

Next, e-Maintenance for FPGA-based system has to be created. HTTP server is to be implemented in order to provide remote connectivity to FPGA platform to allow observing its performance and diagnostic, fixing malfunctions by reloading firmware and upgrading the system by sending a new bitstream over the Internet and reconfiguring FPGA dynamically.

Finally the role of e-Maintenance in periodic partial reconfiguration of FPGA systems based on temporal partitioning for different applications is to be evaluated.

1.3 Project Organization

The project is organized in five chapters. First chapter opens with a brief introduction of the project, stating its objective and organization.

Second chapter follows with theoretical overview of system maintenance in general and e-Maintenance in particular including overview of Internet and its main protocols. FPGA-based system is introduced followed by temporal partitioning and partial reconfiguration. The role of e-Maintenance for FPGA programming and reconfiguration is discussed.

Development of Embedded e-Maintenance for FPGA-based Reconfigurable System is presented in chapter three. Requirements of the system are determined, system architecture is presented and detailed implementation is described.

Chapter four holds experimental results. HTTP server functionality is tested, communication with FPGA platform is established, reconfiguration of FPGA-based system is fulfilled and observation of different FPGA reconfigurations is presented.

Finally, conclusions are drawn in chapter five. Future work and possible further research is discussed.

Chapter Two

Theory overview

2.1 Introduction

In this chapter theoretical overview is presented. It is divided into three major parts:

- system maintenance which talks about system diagnosis and testing process
- e-Maintenance and Internet communication
- FPGA-based system

2.2 System Maintenance

Every system goes through several stages within its lifetime as illustrated in Figure 2.1. In the first stage the requirements of the system need to be determined according to customer's needs.

Based on the requirements specification is prepared. Specification includes functional characterics such as specifying inputs and outputs of the system, operating charateristics such as power and frequency, environmental characteristics such as reliability of the system at a certain temperature range, and so on.

In the next stage all the necessary data for the manufacturing stage is prepared. High-level decisions are made about the overall structure of the system, its architecture, and the strategies used to implement the system. The system is divided into functional blocks and afterwards each block is being designed to details. After the design is complete, the first stage of testing is conducted. It is meant to compare the actual system performance to the required one and in this way to discover the defects of the system, which could be caused by equipment malfunctions, defects in materials and human errors [2].

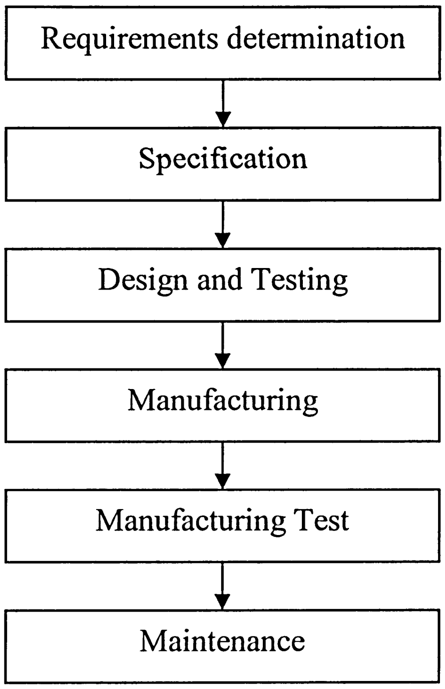


Figure 2.1. System realization process.

In the manufacturing stage mechanical transformation of materials into new products in large quantities is conducted, assembling component parts into new products. After that new products are tested again to verify they are working correctly.

Now the product is ready to be shipped to the customer. Ideally, its performance should be 100% correct for as long as the customer needs to use the product. Unfortunately, the reality is different. At a certain point some defects may appear in the system due to improper handling, environment conditions, aging of the system, or initial defects that were not discovered at the testing stage. Here the maintenance stage comes in, when the supplier must check the system, find the source of the problem and replace the necessary component. If the customer needs to add new features to the existing system, the supplier upgrades it at the maintenance stage as well.

2.2.1 Testing Process

The role of testing is to detect whether there is a defect in the system and if possible to locate it. Basic principal of testing is illustrated in Figure 2.2.

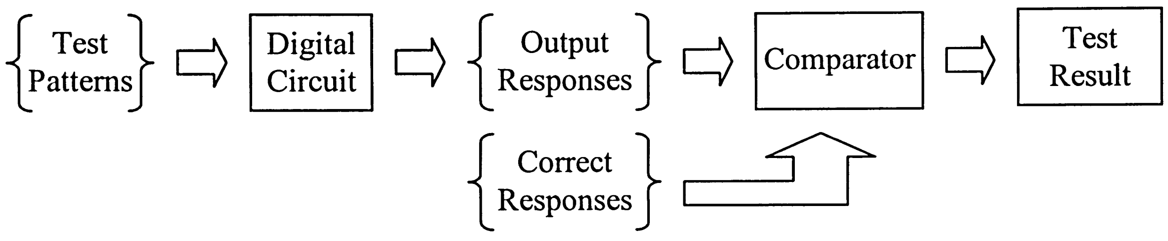


Figure 2.2. Principal of testing.

A test performed on a system may be functional in which case the correct function of the system needs to be verified [2]. A complete functional test will check each entry in the truth table. In case of a large circuit with many inputs such test will take extremly long time to finish. Therefore in most cases only a certain amount of test patterns is applied. Another way to make a large system test less complicated is to divide the system into smaller functional blocks and run an individual test on each of those blocks.

Another type of testing is called structural test. This type of testing depends on the specific structure of the circuit such as gate types, interconnects, etc. These tests enable developing algorithms with the help of fault models. A fault is representation of a defect at function level.

Several levels of fault models are possible:

- Behaivioral level – at this level, which is often called high level, the behaivior of an electronic system is described in computer-readable form and is written in a programming or a hardware description language. The variables and operations correspond to specific application domain and the behaivioral faults refer to incorrect execution of the language constructs. Examples of such faults are branch fault, which affects a branch statement and causes it to branch to an incorrect destination and instruction fault, which causes an instruction to be incorrectly executed. Functional fault models of semiconductor memories are also a part of behaivioral faults, since their function is very simple and exhausive functional test is normally used.
- Register-transfer level (RTL) – or sometimes called logic level, which uses a netlist of gates. Stack-at faults are the most common fault models for this level. Stack-at fault is modeled by assigning a fixed value to a signal line in the circuit. Also common to this level are bridging fault, which represents a short between a group of signals and delay faults.
- Transistor level – often reffered as component level includes stack-open types of faults, which are modeled as the switch being permanently in open or shorted state, considering a MOS transistor as an ideal switch.

With the use of these fault models an algorithm for testing the system can be created. Thus the system can be tested on different levels, from the highest to the lowest.

2.2.2 Maintenance

Maintenance is necessary whenever the system is malfunctioning. During maintenance process the system is tested to determine which part of the system stopped functioning, after which the faulty component is replaced.

Malfunctioning of the system can occur due to various reasons. With the help of latest technologies semiconductor devices operate on a lower power voltages, have smaller sizes and high density. However, these qualities make the system more volnerable to external and internal disturbances such as electromagnetic interference, cosmic radiation and power supply fluctuations [3]. All these disturbances can cause defects in the system.

Defects can be of permanent and temporary nature:

- Hard faults - caused by hardware defect in the circuit and considered to be faults of a permanent nature. Such faults will produce steady system malfunction. Examples of these faults are stack-at faults and stuck-open faults.
- Soft Faults - faults of no permanent nature. These faults happen as a result of external interferences and are called Single Event Upset (SEU). These faults happen occasionally even in case of no electromagnetic interference present and power supply lines being steady, cosmic radiation can still cause SEU.

2.2.3 Diagnosis

Whenever any disturbance occurs, causing the system to stop functioning properly a diagnosis must be performed. Diagnosis is a way to determine exactly what part of the system is corrupted and how to solve the problem with minimum effort and cost. Testing is a very important part of diagnosis.

Even for a relatively small system, consisting of several components, testing can be a tedious process. Running an exhaustive test on a whole system and checking for all possible faults will take an infinite amount of time. In order to perform diagnosis effectively, the system must be divided into blocks and each block must undergo its own separate test. Based on the results of those tests a decision can be made as to which component in the system is faulty.

Diagnosis of a system can be performed on a structural and on a functional levels [4]. On a structural level the division of the system into blocks is performed based on the system structure that is a certain amount of printed circuit boards, chips, wiring etc. The system is divided into blocks according to its components and each of them is tested separately. After the testing is finished the faulty component should be found.

When the diagnosis is performed on a functional level, the testing is based on system functionality. Each system has to perform certain functionality and it can be viewed as a combination of several functional modules. Once the division of the system into these modules is defined, a separate functional test can be performed on each of those modules, thus revealing the faulty one. Functional module corresponds to an actual structural component (or several components), which needs to be substituted. Functional and structural levels may overlap which can create flexibility in a test design. This may allow choosing the type of testing which is easier to perform.

In certain cases diagnosis of communication is necessary. In a complex system where several components are constantly communicating with each other, system malfunctions might be caused by communication faults. One of the tasks of system diagnosis is to detect faults in communication. This can be accomplished with the help of communication protocols, when the transmitting end can send special control messages and the receiver must send appropriate acknowledgments. When the acknowledgement is not received the source of the problem can be in faulty communication or defected component. Communication between components can be improved by adding checksum bytes to transmitted data. These bytes are analyzed by the receiver to verify the integrity of data and if necessary the receiver can send a request to transmit the same data again. Many different encoding techniques are used for verifying data integrity, among which Cyclic Redundancy Check (CRC) is the most popular.

In reloadable systems, which require initialization stage malfunctioning can be caused by faulty elements responsible for uploading the data/software. Usually these elements are memory devices such as FLASH, SRAM, EPROM, etc. A common way to verify correct functionality of such devices is to upload a small amount of data, read it back and compare it with the original. If this test repeatedly results in an error a conclusion can be drawn that the memory device is indeed faulty. With this procedure this single component failure might be verified prior to system initialization stage.

2.2.4 Built In Self Test (BIST)

As technology evolves integrated circuits become more and more complex. Since it is almost impossible to trace and probe internal signals in these circuits it is very difficult to test them thoroughly and quickly. For such systems design-for-test (DFT) techniques are a good solution for improving system testability such as Build In Self Test (BIST) [4]. BIST is very useful for maintenance and diagnosis, since it reduces test generation effort at all levels. This test is specified as one of the system functions. At the highest level of system test, this test is usually implemented in software. Software approach provides flexibility, but on the other hand it can be slow and expensive to develop. Therefore for some systems it is preferable to implement self-testing in the hardware.

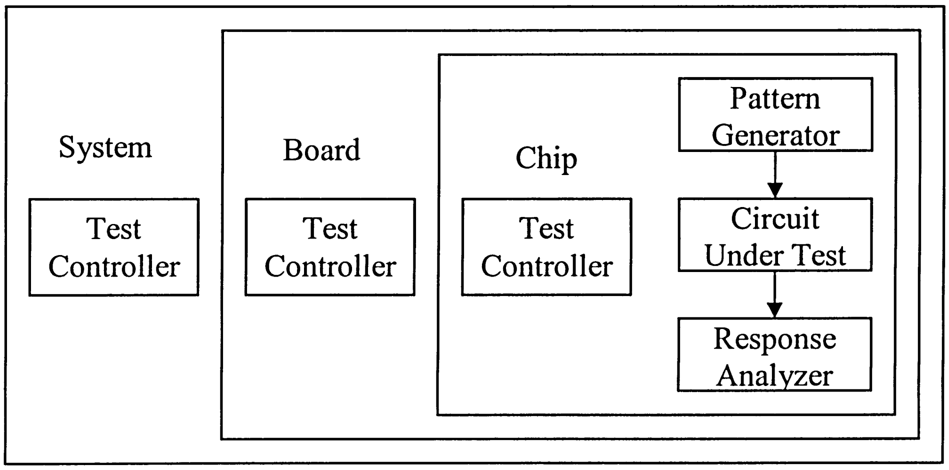


Figure 2.3. BIST hierarchy.

BIST system hierarchy is shown in Figure 2.3. System test controller can activate the test on all PCBs. Each PCB has its own test controller, which in turn activates self-tests on all chips on the board. Likewise each chip test controller activates self-test of the chip itself. After the tests are finished, the results are transmitted to the main test controller: all chips test controllers transmit the results to board test controller, which collects them and sends them further to test controller of the system. Upon receiving all the results, system test controller is able to detect faulty chips and boards and isolate them.

2.2.5 System Monitoring

Real-time continuous monitoring is now generally considered to be a part of the mechanism for improving system reliability [5]. It can significantly reduce the time spent on testing the system, since just by observing the event log system specialist can understand which part of the system is malfunctioning [6]. Also, some malfunctions will only take place under specific conditions, which might not be reproduced in regular off-line testing environment. In order to provide a proper event log, a monitor must systematically observe execution behavior of the system. Its activity must not disturb system functions and it must keep up with target execution by imposing real-time constraints on its internal processing [7]. Such systems are important in general applications that are subject to time-dependent process interaction anomalies.

2.3 E-Maintenance

Effective diagnostics and maintenance are very important factors in system cost, because during the process of diagnostics and repair the system is not functioning. In many cases when a problem occurs to the system, the supplier will send service engineers to customer's site to perform diagnostics, testing, and replacement of faulty components in the system. Sometimes, the service engineer cannot resolve the problem solely and needs to bring the symptom back to the supplier's diagnostics center and uses the information system to inquire the solution information or purchase the required maintenance components. Thus the diagnosis and repair of failed system usually cannot be performed immediately and therefore the downtime of the system can be long and can cause a significant production loss.

With the development of Internet technologies remote connectivity to the system becomes available. This opens huge opportunities for design of remote e-Maintenance systems to perform remote monitoring, remote access to log files and diagnostic, which can significantly reduce the downtime of the system [8].

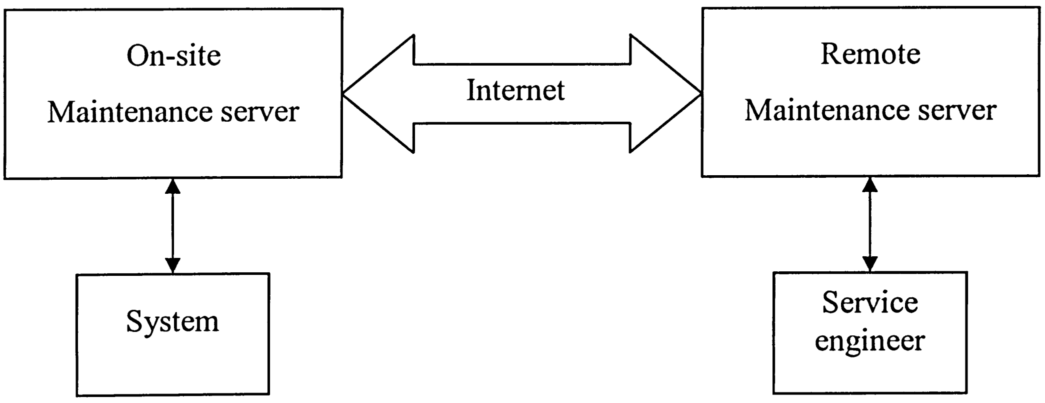


Figure 2.4. E-maintenance system.

Figure 2.4 presents the concept of e-maintenance system. The system preferably containing a self-testing mechanizm and an embedded monitoring device for effective diagnosis is connected by a local network to on-site maintenance server. This server receives log files and self diagnostic results from the system and keeps them in its database, so that they are available for service engineers. Connected to a local host service engineer can access these log files and diagnosis results by connecting to on-site maintenance server through internet. He can view current status of the system, provide diagnostics or even upgrade the system with new firmware without being close to the system physically.

2.3.1 Internet Communication

The Internet is a global system of computer networks. Each computer is independent – a host, and it can get information from any other computer. Internet provides services such as file transfer, email and the World Wide Web. It uses a set of protocols called TCP/IP (for Transmission Control Protocol/Internet Protocol).

Internet communications can be presented with the help of four layers, as illustrated in Figure 2.5.

Application layer	HTTP, FTP, Other Applications
Transport layer	TCP or UDP
Network layer	IP
Data Link layer	Ethernet, Token Ring, Other Network interfaces

Figure 2.5. TCP/IP Reference Model.

Communications functions are divided into responsibilities of each specific layer [9]. These layers are:

- Application Layer – this layer is a client/server communication. Data is sent as commands from the user.
- Transport Layer – is responsible for sending the data, determining whether it has been lost and needs to be resend. Sometimes error check is performed at this level.
- Network Layer – this layer manages the logical connection between two nodes of the network.
- Data Link Layer – at this layer data is transmitted across single network.

The data flow within TCP/IP network is presented in Figure 2.6. Each network layer adds a header to the data, so that the next layer can handle it properly.

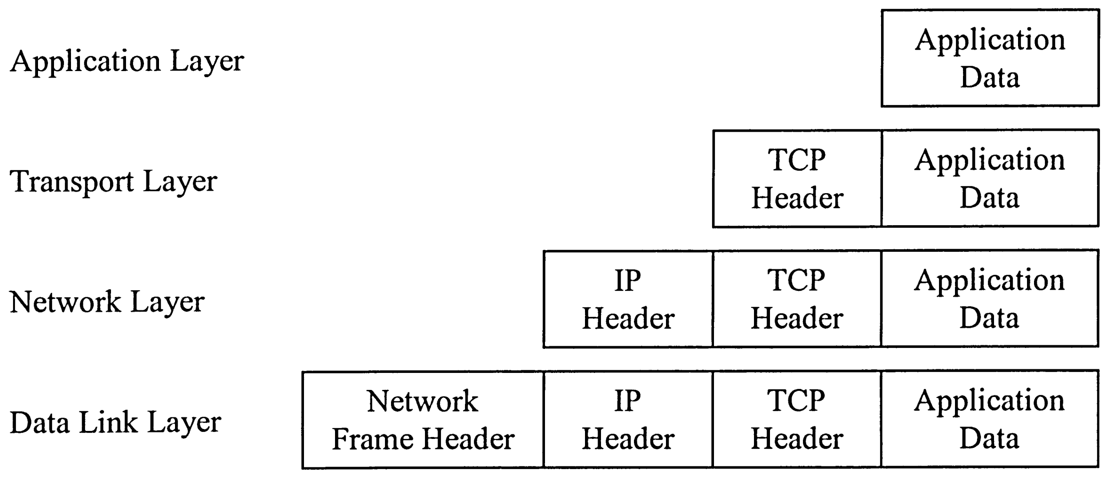


Figure 2.6. Data Flow and Header Utilization within TCP/IP network.

2.3.1.1 World Wide Web

The World Wide Web is a hypertext-based, distributed information system, which provides its users with a graphical interface to access data stored anywhere in the world. Hypertext is a method of creating online documents [9]. Documents written as hypertext contain links to other documents or other resources. Hypertext provides visual indications for these links, such as changing the shape of a mouse pointer when it is placed on a section of a document that points to another resource or highlighting that link. Web documents are created using Hypertext Markup Language (HTML).

A Web server is a program that serves the files from Web pages to Web users on client machines. Web client connects to Web server and sends a request for service. Web servers are used for serving e-mail, downloading requests for File Transfer Protocol (FTP) files, and building and publishing Web pages. Therefore web servers are required to be compatible with application level protocols and TCP/IP protocol (or UDP). Considerations in choosing a Web server depend on requirements of the system and may include how well it works with the operating system and other servers, its ability to handle server-side programming, security characteristics, and publishing, search engine, and site building tools that may come with it.

2.3.1.2 Hypertext Transfer Protocol (HTTP)

Requesting and moving web documents from a server to a client is defined by Hypertext Transfer Protocol (HTTP) [10]. HTTP is a request/response protocol: if the client needs to receive service from the server it must send a request first. HTTP request includes a request method to note what the client needs from the server, what resource the client wants to manipulate through the Uniform Resource Identifier (URI) and a protocol version. A server response also includes the protocol version, followed by a success or an error code. The response also contains server information and web information.

The first HTTP version was HTTP/0.9. This was a very simple protocol for transferring text files only. The next version HTTP/1.0 was a more complicated protocol allowing transfer of many types of files and resources. However this protocol is based on a perception that every time the resource is requested, a new connection must be established. Since web pages became more complex containing several text and graphic files, a new version was developed: HTTP/1.1. This version allows keeping one connection open to send and receive more than one request/ response pair.

Several request methods are used in HTTP/1.1 such as:

- GET – clients use this method to request web content.
- PUT – used by clients to submit material to a specific web URI.
- POST – used by clients to add material to an existing URI.
- OPTIONS – a request for information about available communication options, so that a client can determine the options and requirements of the resource or the capabilities of the server.
- HEAD - used for testing hypertext links for accessibility and recent modification.
- DELETE – this method requests that the origin server deletes the resource identified by the Request-URI.
- TRACE – clients use this method to see what is being received at the other end of the request chain in order to use that data for testing or diagnostic information.

2.3.1.3 File Transfer Protocols

File Transfer Protocols allow the user to access files on remote hosts, send and receive files through Internet [9]. There are two main protocols of such kind: File Transfer Protocol (FTP) and Trivial File Transfer Protocol (TFTP).

With the help of File Transfer Protocol the user can operate on remote resources as if they were on his local computer. The user can view the directories, open files, copy them and move them. FTP is a full-featured protocol, it offers a vast variety of operations between two hosts. Hosts can exchange files regardless of their operating systems or file structure. FTP can handle any type of file and is very reliable. FTP uses two different TCP channels. The first channel is for transferring data such as directory information and transferred files. The second channel is for control purposes, through this channel commands from the client to the server and backwards are sent.

TFTP is a simplified version of FTP. It is useful for hosts with limited memory, limited file download requirement and less need for reliability. It cannot list directory content of another host, its main purpose is to read files from and write files to a remote server. TFTP uses UDP and therefore files are transferred through independent exchange of packets. The client sends an RRQ (read request) or WRQ (write request) packet to the server, containing the filename and transfer mode. The server replies with an ACK (acknowledgement) packet and sends the first part of the file. The client sends an acknowledgment upon receiving the packet, and the server sends the next part. This process continues until the whole file is sent. TFTP protocol is very simple, it has only five types of messages: Read request, Write Request, Data to be read or written, Acknowledgment and Error.

2.3.1.4 Transport Layer Protocols

Transport layer provides interface between a network layer below and an application layer above. There are two primary transport layer protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) [9].

TCP is a connection-oriented protocol, which means that the data can be transmitted from one host to another only after both hosts have initiated a connection. This requires a handshaking process called three-way handshake to be completed before establishing the connection. One end is sending a request to the other host to open a TCP circuit. The client-side of a connection sends an initial SYN segment to the server to show that the synchronization process is taking place. Upon receiving the acknowledgment the server-side should respond to a valid SYN request with a SYN/ACK. Finally, the client-side should respond to the server with an ACK, completing the 3-way handshake and connection establishment phase.

TCP connection is duplex, meaning hosts can send data at the same time and it will travel in parallel. Also TCP is a reliable delivery protocol – it guarantees delivery of data between two hosts. Both hosts must acknowledge of all data received from the other host. In case of missing data it will be retransmitted.

TCP Header Structure is shown in Figure 2.7.

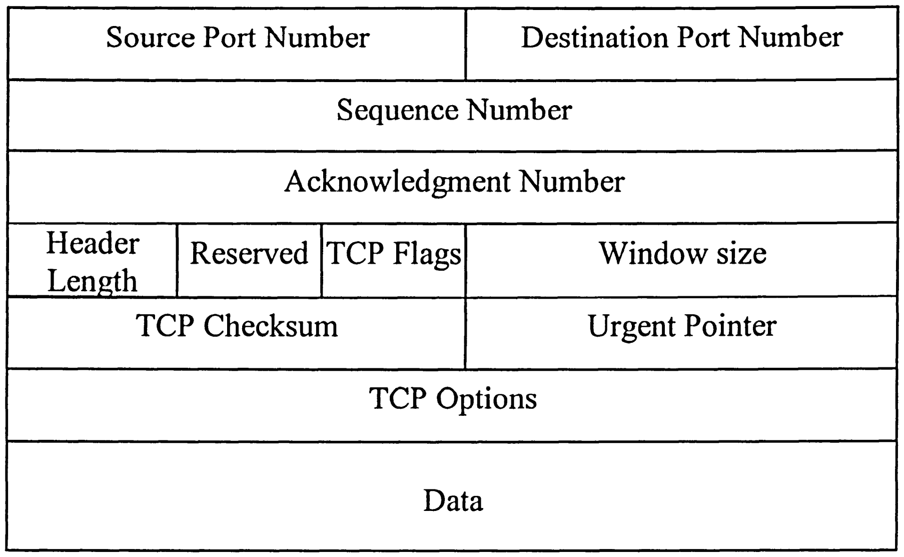


Figure 2.7. The TCP header.

- The first two fields are source and destination port numbers. They are used by the receiver and the transmitter respectively to identify each other.
- Since data is divided into packets rather than sending the whole chunk at once, sequencing is necessary. Sequence number ensures that packets will not be misordered and missing packets will be noted.
- Header length field is usually 4 bits in length, but since inclusion of options can result in variable-length header, the presence of this field is necessary.
- TCP flag bits are used to negotiate and manage the connection.
- Window field indicates the maximum number of bytes for the receiver to expect.
- Checksum field provides error detection capability.
- Urgent pointer field is used when the URG flag in TCP flags field is set indicating that current TCP segment is urgent.

- TCP options field enables the support of various options such as Maximum Segment Size.

Unlike TCP, UDP is a connectionless protocol, which means that an application using UDP can have its data transmitted without establishing a connection with the receiver first. This simplifies the communication process. UDP does not provide the reliability and ordering guarantees that TCP does; messages (called datagrams) may arrive out of order or go missing without notice. However, as a result, UDP is faster and more efficient for many lightweight or time-sensitive purposes for which using TCP would result in a high level of overhead. Also UDP header is simplified and is much smaller than that of TCP. It is presented in Figure 2.8.

Source Port Number	Destination Port Number
UDP Datagram Length	UDP Checksum
Data	

Figure 2.8. The UDP header.

- Source and destination port numbers are used by the receiver and the transmitter respectively to identify each other.
- Length field indicates the length of UDP datagram.
- Checksum field is used when checksum is required by the application.

2.3.2 E-Maintenance of Reconfigurable Systems

Previous generation of E-maintenance systems dealt with embedded microprocessors based systems. It was possible to conduct remote monitoring and diagnosis for them, thus internet was merely servicing information exchange such as log files or testing results. Today, as FPGA's become more and more popular and many systems are based on them, E-Maintenance can have a new important task – remote repair of the system through dynamic reconfiguration of processor structure. Thus without substituting the hardware remote reconfiguration of the system for repair as well as for new applications is becoming possible.

2.4 FPGA-based System

FPGA stands for Field Programmable Gate Array. It is a digital integrated circuit that contains programmable logic blocks and configurable interconnects between them [11]. Figure 2.9 illustrates generics FPGA logic and routing resources.

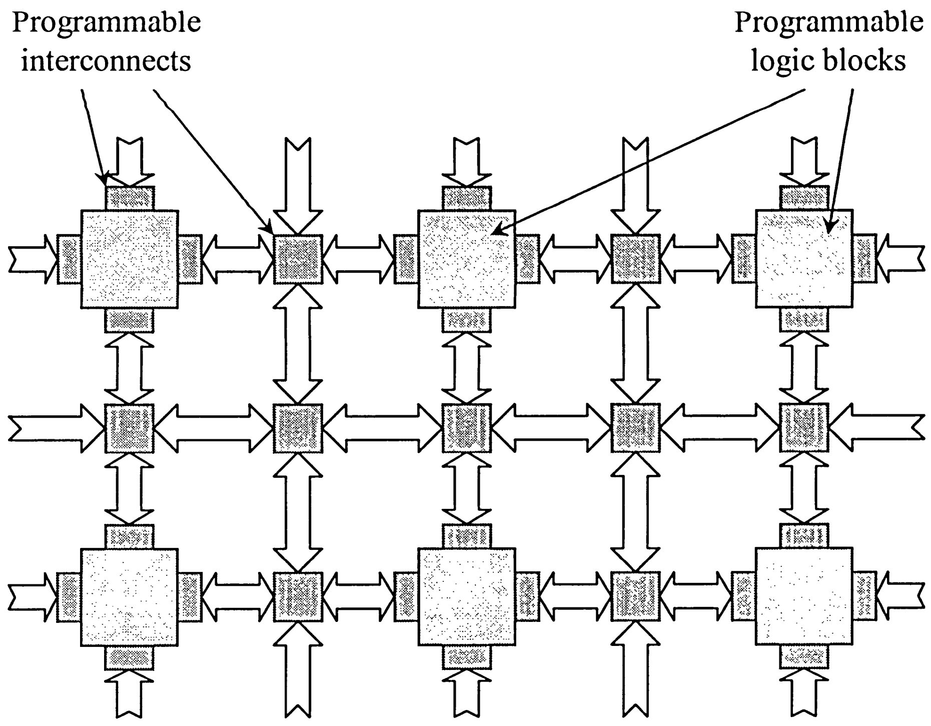


Figure 2.9. Generic FPGA logic and routing resources.

Based on their implementation FPGAs can be reconfigured multiple number of times, while other type of FPGAs can be programmed only a single time.

Today FPGAs are used to implement a variety of products such as communications devices, digital signal processing applications and embedded microcontroller applications. FPGAs are being used to implement designs that previously have been implemented using application-specific integrated circuits (ASICs). Although the cost of

ASIC in massive productions is lower than that of FPGA, implementing design changes is much faster in FPGA as well as the time to market of FPGA design is faster. These qualities make FPGA very popular, especially for small innovative design companies.

2.4.1 FPGA Architecture

The main feature of FPGA is its ability to be reconfigured many times. In this section the mechanism that allows reprogramming FPGAs is explained as well as some architectural features.

2.4.1.1 LUT-based logic blocks

Lookup table (LUT) based blocks consist of a multiple input lookup table, a register that can act as a flip-flop or a latch and a multiplexer, as shown in Figure 2.10.

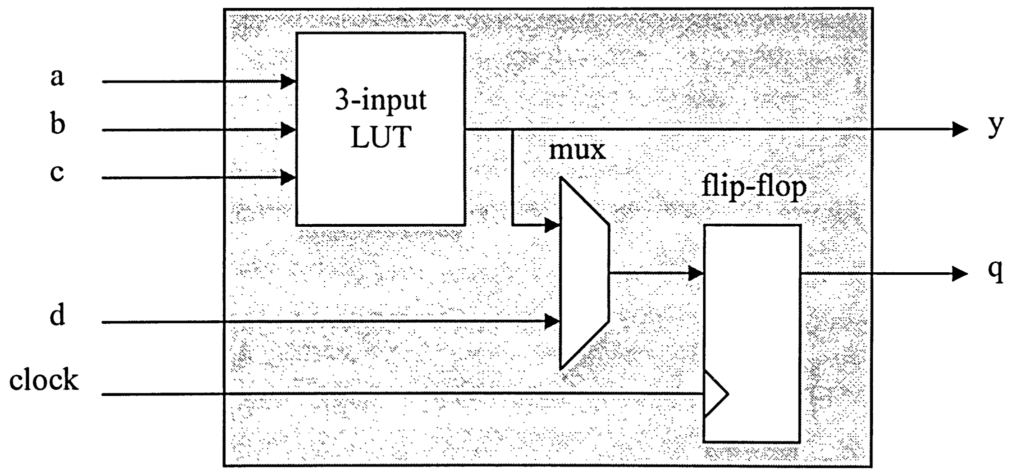


Figure 2.10. A simplified view of programmable logic block.

Each one of those blocks can be configured to perform a different function [11]. For example, if LUT is required to perform the function y = (a OR b) AND (NOT c). The

LUT needs to be loaded with values as shown in Figure 2.11. 8-1 Mux represents a cascade of transmission gates for selecting the desired SRAM cell.

Truth table

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

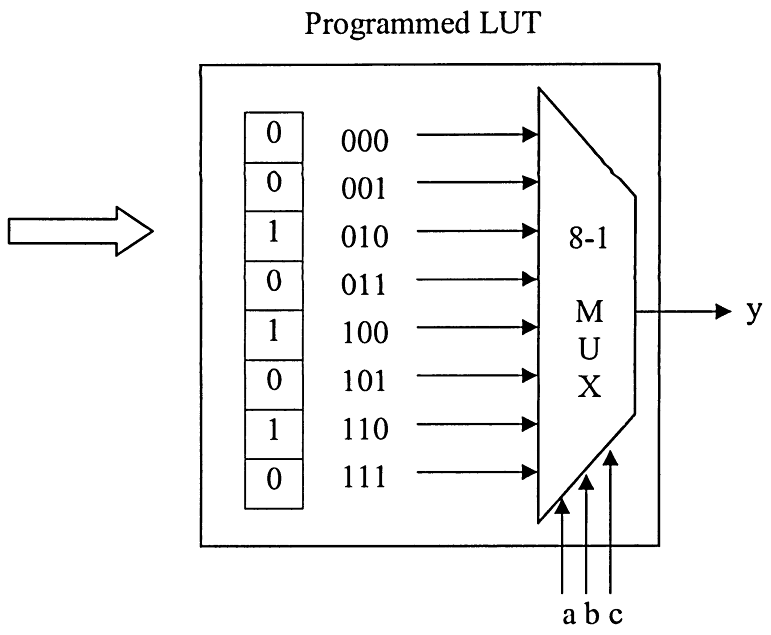


Figure 2.11. LUT configuration.

Different number of inputs has been researched in the past. Today all successful architectures use 4-input LUTs.

2.4.1.2 SRAM-based Technology

The majority of FPGAs are based on the use SRAM configuration cells [11]. Static RAM is employed because the values in the cells don't need to be refreshed as in dynamic RAM. They remain unchanged unless altered by designer or until the power is removed from the system. SRAM cell consists of multitransistor SRAM storage elements. Output of each element drives an additional control transistor. Based on element's content (0 or 1) this control transistor will either be OFF or ON.

SRAM-based FPGA has to be reprogrammed every time the system is powered on, which requires the use of an external memory devise. Since each cell consists of several transistors, it consumes a lot of silicon. However, SRAM-based technology's big advantage is that these cells can be reprogrammed quickly and repeatedly.

2.4.1.3 Embedded RAMs

Since many applications require the use of memory, latest FPGAs include embedded RAM called block RAM [11]. These blocks can be positioned around the periphery of the devise or organized in columns, as shown in Figure 2.12.

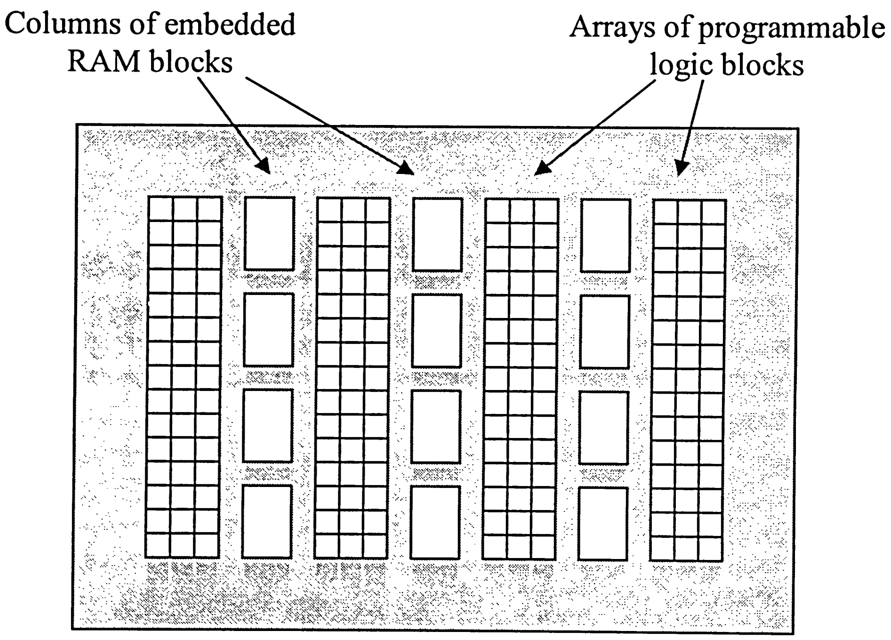

Columns of embedded RAM blocks
Arrays of programmable logic blocks

Figure 2.12. Simplified view of chip with embedded RAM blocks.

Each block of RAM can be used independently or if a larger chunk of memory is needed, several blocks can be combined to be used as one. They can be used to implement different functions such as FIFO, state machines and so on.

2.4.1.4 Embedded processor cores

Most of electronic designs are implemented using microprocessors. Until recently microprocessors have been included in the design as a separate device. Lately they became available as part of high-end FPGAs [11]. These microprocessors are referred as embedded cores. Using such FPGA saves the cost of having two separate devices, as well as makes the design simpler, since there is no need in connecting those devices on board. Microprocessor core can be implemented in FPGA in two ways: as hard core and soft core.

A hard microprocessor core is a predefined block in FPGA, positioned at a certain place in the chip. This place can be on the side of the main FPGA fabric called "strip", as shown in Figure 2.13. Additional functions can be added with microprocessor core such as memory, peripherals, etc. More than one microprocessor can be embedded into the chip.

Another option is for embedded core to be placed directly into the main FPGA fabric. In this case the memory necessary for the core is formed from embedded RAM blocks, and various peripherals are formed from programmable logic blocks. The advantage of this position is speed gain due to proximity of microprocessor core to the main FPGA fabric.

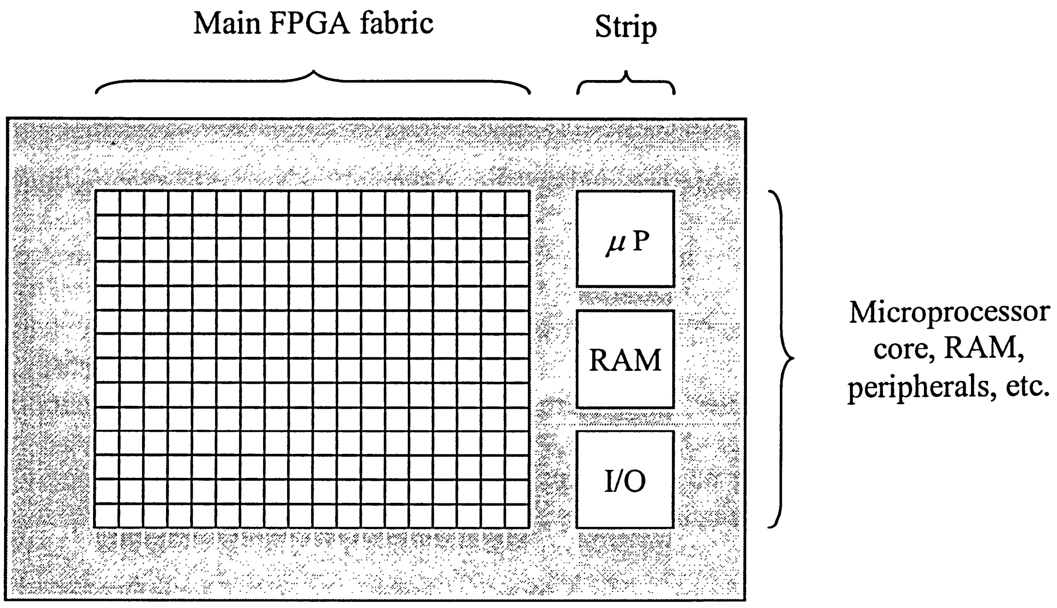


Figure 2.13. Chip with embedded core outside of the main fabric.

Soft cores are programmed into FPGA by configuring a group of programmable blocks to act as a microprocessor. Such core can be provided in a form of RTL netlist that will be synthesized with the other logic or as a placed and routed block of LUTs. All additional devices like memory controllers, interrupt controllers and so on are implemented in the same way as microprocessor core.

These cores are simpler and up to 50% slower than hard cores. However they give the designer the flexibility to include the core according to his needs as well as any peripheral device. Also several cores can be implemented as long as there are enough programmable blocks in FPGA.

2.4.2 Diagnosis of FPGA-based system

SRAM-based technology is very sensible to Single Event Upsets (SEU). Such events may be originated by high energy particles hitting the silicon substrate, and thus changing the logical state of the memory elements [12]. SEUs may alter the memory elements in the design such as the content of a register in the data-path, or the content of the state register of a control-unit. Another aspect of SEU effects is possible change in the content of the memory storing the device configuration information, such as the content of a Look-Up Table (LUT) inside a Configurable Logic Block (CLB) or the routing of a signal in a CLB or between two CLBs. Therefore SEU by changing one bit can cause malfunctioning of the whole device.

One way to avoid this problem is to use radiation hardened FPGA devices [13]. However, these devices are much more expensive. Another possible solution is to use FPGAs based on antifuse technology, but they have a major drawback – they cannot be reprogrammable.

Therefore periodic testing of FPGA is necessary. It will detect the effect of SEU and by reprogramming FPGA the defect can be fixed. FPGA-based systems which are designed to work in outer space are equipped with a special reloading system. This system allows to reprogram FPGA frequently while FPGA is in working mode. Such systems are very expensive, but necessary in space conditions with very high ratio of SEU.

It is also important to test FPGA regularly because some manufacturing defects in the reconfigurable hardware may result in faulty behavior only for spesific configurations. The malfunctioning FPGA component may actually pass manufacturing test and work properly for a number of different configurations, but one particular configuration might cause it to fail [14].

E-maintenance opens new possibilities for repair of FPGA-based systems, since whenever FPGA needs to be reconfigured, new firmware can be sent to the system remotely via internet.

2.4.3 Partial reconfiguration of FPGA

Not only can FPGA be reprogrammed with new logic whenever it is necessary, but it can also be reprogrammed partially [15]. A certain predefined subset of FPGA can be reconfigured, while the rest of the device continues operating. The process of partial reconfiguration is based on dividing FPGA into blocks according to their functionality and reloading each block with it's own bit stream. Thus in order to reconfigure a part of FPGA, it is not required for the whole system to stop operating. This feature comes very handy for systems that deal with mission-critical tasks that cannot be interrupted.

2.4.3.1 Run-Time Temporal Partitioning of FPGA Resources

Today digital signal processing systems need to process streamed data at very high speed. Traditionally embedded microprocessors were the basis for designing such systems. However, the performance of such systems is limited due to sequential process of instructions execution in microprocessors. Furthermore, microprocessors require many additional clock cycles for each instruction: instruction fetch, instruction decode, data fetch, store result. Thus completion of a certain set of operations will take longer time than just data processing itself.

For example, suppose the following set of operations is to be implemented:

$$Z_i = A_i + B_i \times C_i, \text{ for } i = 1, 2, \ldots, 100$$

For each value of vector elements 7 instructions have to be executed: load B, load C, multiply B and C, load A, add A to the result of previous operation, increment i, compare i to 100. Even if we assume that the number of clock cycles for each operation is 1, the whole task will take 7 clock cycles * 100 = 700 clock cycles.

But if this function is to be implemented in a specially designed circuit presented in Figure 2.14 it will take far less time to execute: Total time = Latency + Output rate * 1 clock cycle = 2 clock cycles + 100 * 1 clock cycle = 102 clock cycles. That is 7 times faster. In reality multiplication and adding take much more time than 1 processor clock cycle, therefore custom designed circuits operate even faster than shown in this example.

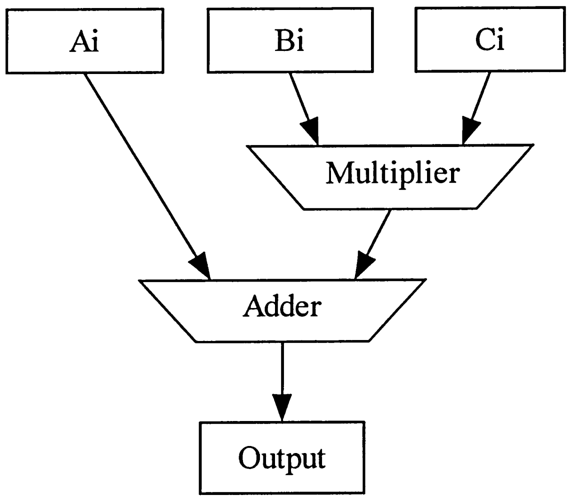


Figure 2.14. Stream processing circuit for $Z_i = A_i + B_i \times C_i$

Growing demand for high-speed data processing resulted in moving away from microprocessors to implementing such systems in ASIC or FPGA devices. As was noted previously, ASIC has major drawbacks compared to FPGA, since FPGA has a lower time-to-market and can be reprogrammed. Furthermore, use of partially reconfigurable FPGA devices makes possible achieving high-cost efficiency. This is done by dividing FPGA into segments when each segment is configured with a specific IP-core designed for a certain task. After the task was executed, temporal results are stored in the memory and a new configuration data reloads FPGA for the next portion of operations to be executed. Such approach allows using same resources of FPGA for execution of different tasks during different time periods and as a result minimization of hardware per task. Figure 2.15 illustrates the concept Run-Time Temporal Partitioning when S_1, S_2, ..., S_i are segments.

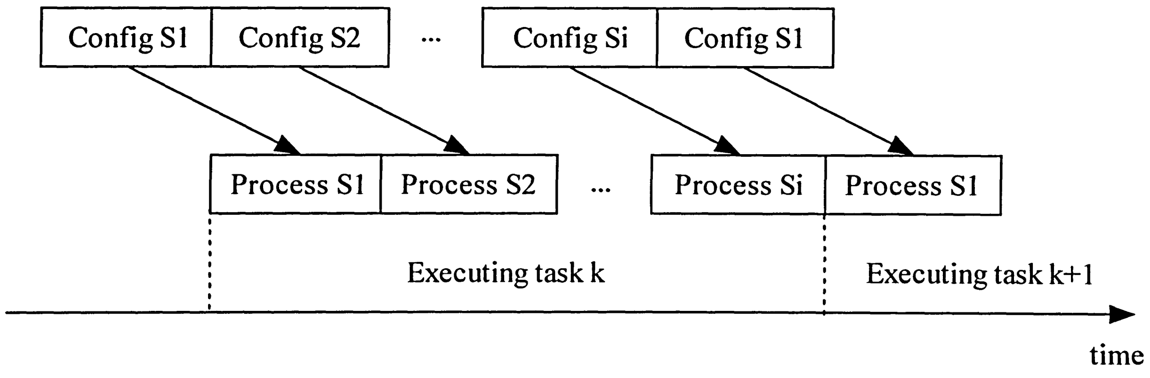


Figure 2.15. Run-Time Temporal Partitioning.

2.4.3.2 Macro-Programming of FPGA-based System

Utilization of FPGAs can be quite complicated. It is expensive and requires a lot of time since many stages are involved: development, circuit design, prototyping and verification. If a complex system is implemented it can result in using a large and therefore extremely expensive FPGA due to complex routing inside the chip. As an outcome a lot of FPGA's logic resources will not be used for the processing itself. Also it will require long compilation cycle. And finally, FPGA prototyping calls for highly qualified developers. Such developers are hardware designers and usually are not experts in the technological process itself.

A concept of Macro-Programming presented in [1] investigates the possibility of simplifying task programming without detailed knowledge of FPGA hardware organization. It is based on dividing the stream-processing application into segments S_1, S_2, ..., S_i when each segment is linked to a given stream-processing unit configured in FPGA. Each of those stream-processing units is called virtual hardware component (VHC) and functions like IP-core, it is used for configuration of FPGA for a specific task – macro-operator, it has interface to memory units for temporal data storage and possible Input/Output interface. Assuming a library of VHC-cores can be produced that for every category of applications, such library is to be supplied by vendors of FPGA platform. Each VHC should be optimized for a certain task and associated with a specific macro-operator. A sequence of macro-operators scheduled according to sequence of tasks is called Macro-Program. Figure 2.16 illustrates the concept of using Macro-operators for application programming.

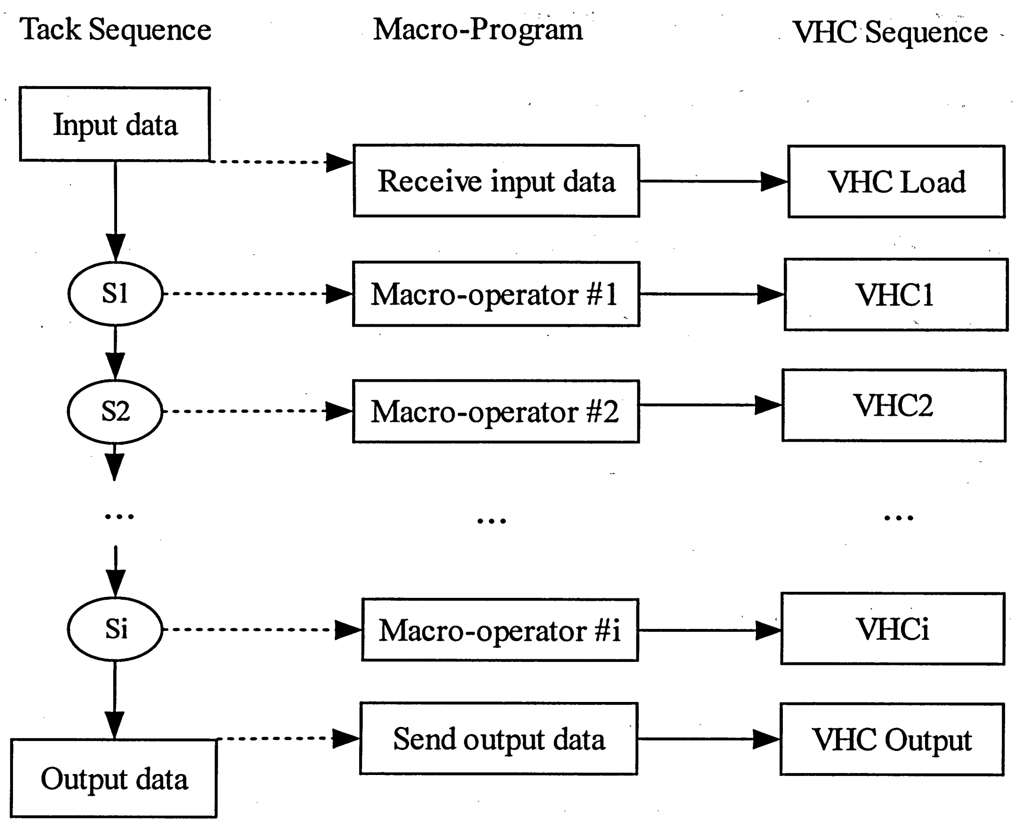


Figure 2.16. Utilization of Macro-Operators.

Such approach allows easier task programming since first of all breaking tasks into segments simplifies the development of the system. Each separate segment is less complicated thus it is easier to design a stream-processing circuit for its fulfillment. Second, macro-operators could be used for execution of different tasks and this way a library of macro-operators can be created. And finally, since the stage of hardware implementation will be finished with the development of stream-processing circuits, task programming itself becomes easy for the programmer. He needs to know how to operate with macro-operators, that is parameterize them correctly and sequence them according to the specific technological process. Eventually, the program will be a schedule of macro-operators, loading appropriate bit-streams into FPGA in specified time periods.

Such program can be loaded to any system containing FPGA-based stream-processing platform. E-maintenance system opens number of possibilities for Macro-programming. The whole Macro-Program can be uploaded remotely. A library of Virtual Hardware Components can be developed remotely and uploaded via Internet as well. If later on this library has to be extended or updated, necessary VHC can be developed remotely and loaded via Internet. It can then be used by the Macro-Program for FPGA reconfiguration for updated task. Thus the system can be upgraded without stopping the execution of its other tasks.

2.5 Summary

A theoretical overview of e-Maintenance and FPGA was presented in this chapter. As part of e-Maintenance general system maintenance was described and overview of Internet communication with main Internet protocols was demonstrated. Theoretical background of FPGA was presented as well as latest possibilities for its reconfiguration were studied.

Chapter Three

System Development

3.1 Introduction

In this project a prototype of e-Maintenance system for FPGA-based reconfigurable system will be implemented. Requirements for such system, its architecture and implementation are described in this chapter.

3.2 Functional Specification

An e-Maintenance system for FPGA-based reconfigurable system must possess the following functionality:

- Provide remote connectivity to supporting personnel via Internet.
- Capable of receiving new firmware via Internet.
- Allow reloading FPGA's firmware.
- Provide monitoring of reconfigurable system and keep log files of system events.
- Provide diagnostic of reconfigurable system.
- Allow viewing log files and reconfigurable system performance via any standard browser.
- Allow downloading log files via Internet.

3.3 Basic Requirements of the System

In order to possess all the capabilities listed above this system must fulfill several basic requirements. First of all it must contain an embedded web server to enable remote connectivity, viewing log files of reconfigurable system, downloading them, and sending new firmware if necessary. System maintenance web page presenting system performance, faults and diagnostics must be designed. The system upgrade and downloading log file will be accessed through maintenance web page.

Therefore the requirements of a web server are:

- To have one or two simple web pages.
- Provide uploading/downloading files capability.
- Provide TCP/IP connectivity.
- Provide compatibility with HTTP protocol.

Second, the system must have enough memory to store log files, embedded web server related items such as web pages, controls, etc.

Finally, an embedded microcontroller is necessary to coordinate the whole system. It will monitor reconfigurable system performance, maintain system log file and provide boot loading to the system.

And like general embedded systems this system should be flexible, compact, low power and low cost to be affordable for mass production.

3.4 System Architecture

The overall block diagram of the system is presented in Figure 3.1.

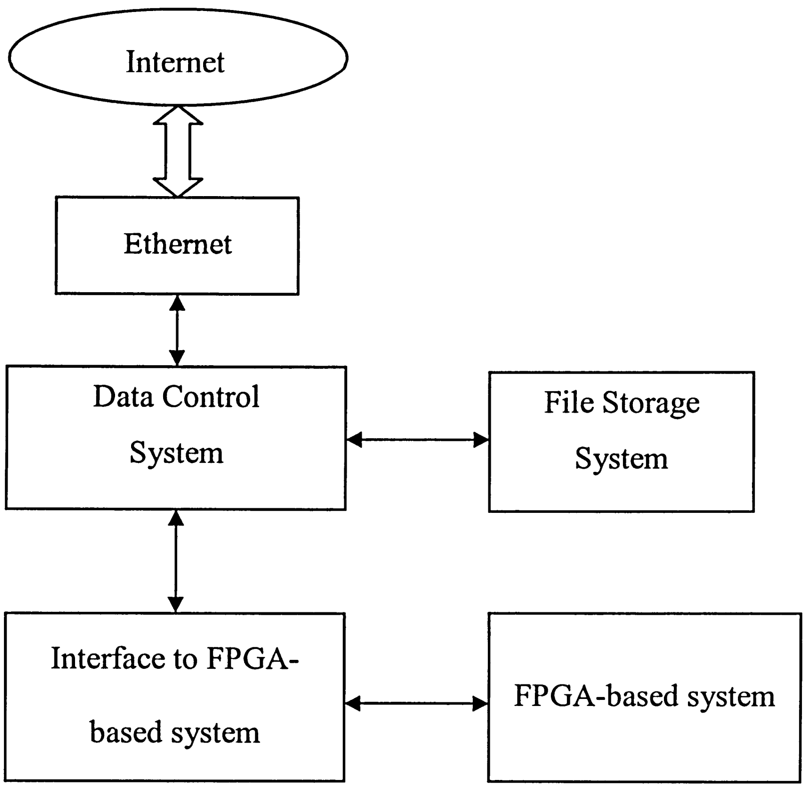


Figure 3.1. Overall system block diagram.

The architecture of e-Maintenance system will be based on choosing a suitable microcontroller and implementation of TCP/IP protocol. For the needs of this system an 8-bit microcontroller will be suitable since it combines sufficient functionality and peripheral devices with less complexity and low cost.

Today there is a variety of available commercial technologies for implementation of network-enabled devices [16]. There are several microcontroller boards on the market that claim to be web-enabled while others can be built from a schematic and a few components. Among such boards are PicoWeb using Atmel's AT90S8515 microcontroller [17] and HTTP, TCP/IP protocols, OT731 from Orlin Technology using Microchip PIC16F877 microcontroller [18] and TCP/IP, UDP, PPP protocols [19], PowerCore Flex [20] using Rabbit 3000 microcontroller [21] and HTTP, TCP/IP, FTP protocols. Many of these devices are simply embedded web page servers, but for the purposes of this project a microcontroller with a potential for a wider range of applications is necessary. Convenient and available development tools and functionality of protocols of different layers of Internet are also important. Based on all these criteria PowerCore Flex from Z-World Inc. is selected as the platform for this project.

3.5 Hardware/Software Partitioning

Hardware and software partitioning are based on the project architecture and defined by the chosen platform for the project.

The hardware in this case is as follows:

- 8-bit microprocessor.
- Static RAM for data.
- Flash memory for instructions storage.
- Serial Flash memory for file storage.
- Programming port for software development and debugging as well as hardware for connection between the development environment and the board.
- Ethernet port.
- Serial ports for communication with serial flash and interfacing with system under test.

- Serial cable with custom RS-232 headers for communication with system under test. Since the RS-232 header on PowerCore prototyping board is not standard, a custom cable for correct pinout had to be prepared.

PowerCore 3800 comes with Dynamic C which is an integrated development system specially designed for programming embedded systems [22]. It provides a built-in full-featured text editor, several quick compiling options and a number of debugging features for an easy development of embedded software. Dynamic C comes with many function libraries to support real-time programming, machine level I/O and TCP/IP stack. Dynamic C implementation of TCP/IP consists of several libraries [23]. These libraries handle application layer protocols, such as HTTP and FTP, transport layer protocol and Internet layer protocol.

The following things will be implemented in software:

- HTTP web server.
- TCP/IP protocol.
- Communication with system under test.
- Project-specific application program.

Application program will establish connection with remote web client, process commands, control the uploading of FPGA firmware and downloading of log files, manage the storage of those files in external memory, monitor reconfigurable system and update log files. Implementation of TCP/IP is a set of Dynamic C functions, which have to be called by application program to deal with protocols, Internet connection, etc. It has to interface properly with application layer and that is a responsibility of application program as well.

3.6 System Hardware Components

Figure 3.2 shows PowerCore FLEX subsystems based on Rabbit3000. Like most typical embedded systems, it consists of CPU, program memory, data memory, crystal oscillator and voltage level converter [24]. PowerCore FLEX also contains the Ethernet port, which comes handy for this project implementation.

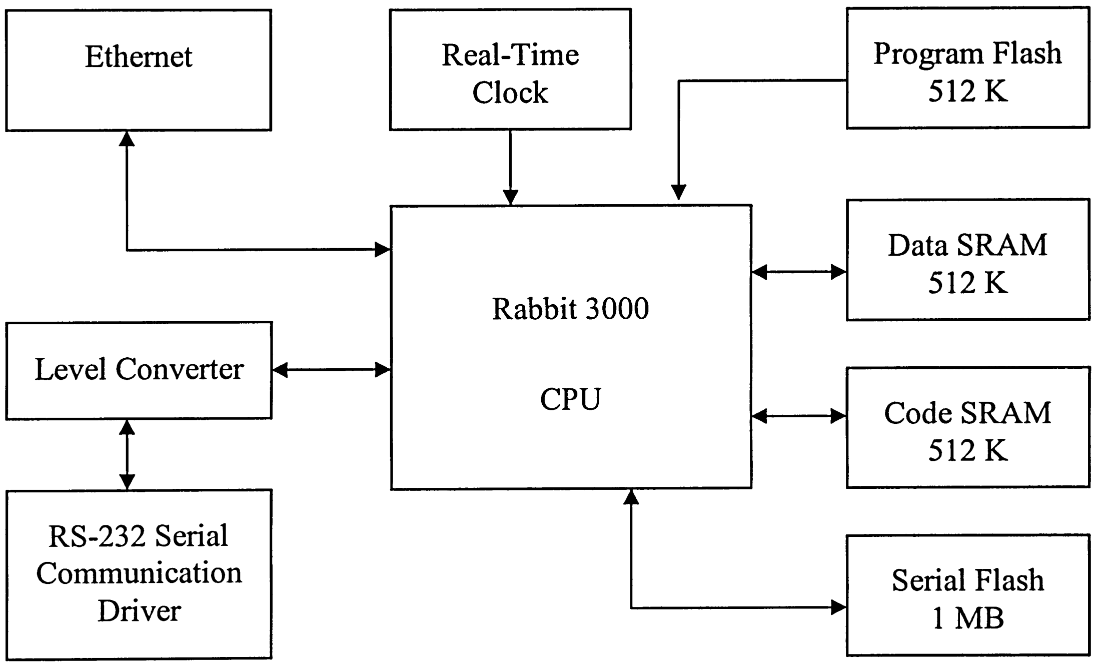


Figure 3.2. PowerCore Subsystems.

Detailed schematics of PowerCore module as well as PowerCore Prototyping board are attached in the appendix A1 and A2.

3.6.1 Serial Communication

3.6.1.1 Serial Ports

Rabbit 3000 has six serial ports – A, B, C, D, E and F [21]. In the implementation of this system three ports are being used:

- Serial port A is used as a programming port.
- Serial port B is used for communication with serial flash.
- Serial port E is used for communication with system under test.

Serial port signals are presented in the following table:

Table 3.1. Serial Port Signals.

Serial Port	Pin Name	Signal Name	Function
Serial Port A	PC6	TXA	Serial Transmit Out
	PC7	RXA	Serial Transmit In
	PB1	CLKA	Clock for clocked mode (bi-directional)
	PD6	ATXA	Alternate serial transmit out
	PD7	ARXA	Alternate serial receive in
Serial Port B	PC4	TXB	Serial Transmit Out
	PC5	RXB	Serial Transmit In
	PB0	CLKB	Clock for clocked mode (bi-directional)
	PD4	ATXB	Alternate serial transmit out
	PD5	ARXB	Alternate serial receive in
Serial Port E	PG6	TXE	Serial Transmit Out
	PG7	RXE	Serial Transmit In
	PG4	TCLKE	Optional external transmit clock
	PG5	RCLKE	Optional external receive clock

3.6.1.2 RS-232

PowerCore Prototyping board has RS-232 serial channel, which is connected to RS-232 transceiver chip [20]. It provides voltage output required meeting the RS-232 standard, in other words, it translates Rabbit 3000 voltages to RS-232 signal levels. Serial Port E signals TXE and RXE are connected to RS-232 transceiver, thus communication with system under test is done through this port. Figure 3.3 illustrates the interface to RS-232:

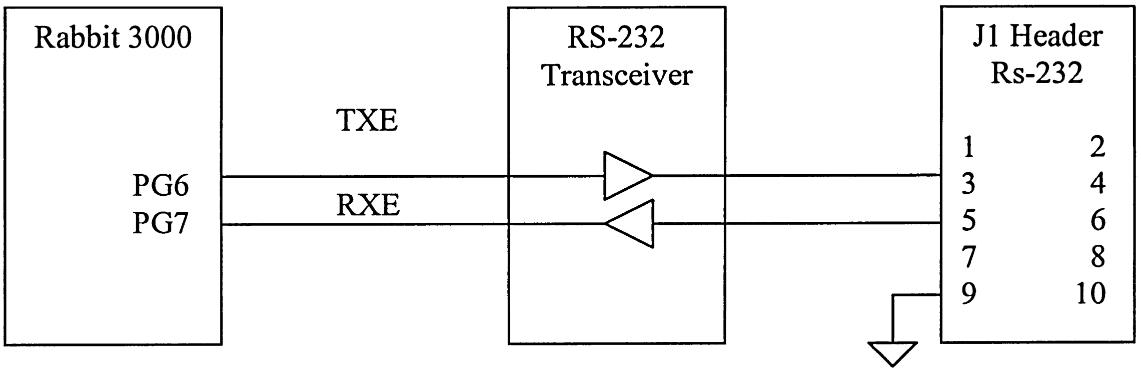
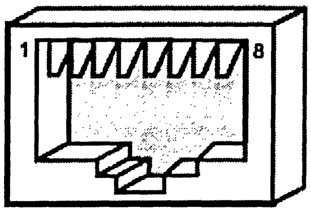


Figure 3.3. RS-232 Interface.

3.6.1.3 Ethernet Port

Finally, there is an Ethernet port for remote connectivity via Internet. PowerCore FLEX module uses a 10/100-compatible 10Base-T Ethernet interface [20], which is the most common scheme. Whether it is connected to a hub or Ethernet adapter, they can be a 10 Mbps unit, a 100 Mbps unit, or a 10/100 Mbps unit.

The RJ-45 connector is similar to U.S. style telephone connectors, except it is larger and has 8 contacts. Ethernet port pinout is presented in Figure 3.4:



1. E_Tx+
2. E_Tx-
3. E_Rx+
6. E_Rx-

Figure 3.4. RJ-45 Ethernet Port Pinout.

3.6.2 Memory

1 Mbyte serial flash is available for data storage [24]. It is used for storing web server related items such as web pages, forms, etc. It can also be used for log file storage – once the log is received from system under test, it has to be saved somewhere, so that it can be made available for remote user.

3.7 System Software Implementation

Dynamic C has various libraries for implementing different layer protocols. HTTP protocol is implemented by HTTP.LIB, and TCP/IP implementation is done in DCRTCP.LIB [23]. Therefore these libraries should be included in the application program and this is done with the help of the following macros:

```
#use "dcrtcp.lib"

#use "http.lib"
```

3.7.1 TCP/IP Stack Initialization

TCP/IP initialization is done by calling the function `sock_init()` [23]. This function takes care of the following:

- Subsystem initialization for TCP
- Initializing the packet driver
- Clearing router and other server tables
- Providing required delay for Ethernet initialization

3.7.2 Interface Configuration

Dynamic C has a set of predefined configurations in `tcp_config.lib` [23]. The first configuration is going to be used. It is a simple configuration of single Ethernet interface. Thus the following macro is necessary:

```
#define TCPCONFIG 1
```

3.7.3 Allocating Socket Buffers

There is a specific macro to define the number of sockets with preallocated buffers that can be used successfully:

```
#define MAX_TCP_SOCKET_BUFFERS 1
```

Set this number to 1.

Next is to define TCP buffer size:

```
#define TCP_BUF_SIZE 2048
```

3.7.4 Specifying a Listen Queue

In order to allow handling of multiple HTTP requests the following function is used [23]:

```
tcp_reserveport(80)
```

With this function a TCP port number 80 which is well known port reserved for HTTP is specified to have pending connection queue which allows it to have many incoming connection requests. New connections will be held in queue until current active connection is terminated. Without this option incoming connection will be cancelled if the socket is already in use.

3.7.5 HTTP server implementation

HTTP server allows clients access its resources such as HTML pages via web browsers and serves clients requests through those pages. It is implemented by HTTP.LIB and it requires setting up the network subsystem as was described in previous sections.

3.7.5.1 HTML page

In order for the HTML page to be available to the clients all of its contents including text and image files is stored in the server's memory. When the server receives a request from the web browser referring to a specific web page, it looks up the name, opens the page and sends the content back to the client. The web page is imported into flash memory by using the `#import` directive:

```
#ximport "Project/HTML/emaintenance_server.html" index_html
```

`index_html` indicates the start of the file for the server to know where to get it.

3.7.5.2 MIME Type Mapping Table

The server has to present to the web browser how the content of the web page is to be presented to the user. This is done in MIME (Multipurpose Internet Mail Extensions) type mapping table which relates common PC-type file extension values to recognized MIME types and vice versa. In the code it is done as follows [23]:

```
SSPEC_MIMETABLE_START
        SSPEC_MIME(".htm", "text/html"),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
        SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END
```

3.7.5.3 CGI Functions

In order to serve user requests HTTP server invokes CGI functions. CGI (Common Gateway Interface) is a standard for interfacing external applications with information servers. Thus, when the user retrieves a specified resource through the web browser, HTTP server calls an appropriate C function to generate a response to the user's request. This function is executed in real-time, and it can generate web page content dynamically like causing the browser display information to the user as well as reading data that was send by the browser.

In the HTML page actions that can be requested to be performed by the user are presented as follows:

```
<FORM ACTION="upload.cgi" METHOD="POST"
enctype="multipart/form-data">
<INPUT TYPE="SUBMIT" VALUE="Upload"> </FORM>
```

When the user clicks the Upload button, a C function referred to upload.cgi action is called.

3.7.5.4 Resource Table

When a user requests a specific action to be performed HTTP server needs to know which CGI function to invoke. Information relating CGI functions and action parameters in web pages is held in resource table [23]. It refers all the URLs to the resources on the server and is defined as follows:

```
SSPEC_RESOURCETABLE_START
     SSPEC_RESOURCE_XMEMFILE("/index.html", index_html),
     SSPEC_RESOURCE_CGI("upload.cgi", upload_cgi),
     SSPEC_RESOURCE_FUNCTION("/download.cgi",
     download_cgi),
     SSPEC_RESOURCE_FUNCTION("/DeleteSlot.cgi",
     delete_slot_cgi),
SSPEC_RESOURCETABLE_END
```

Thus, when Upload button is clicked, upload_cgi function is called.

3.7.5.5 HTTP structure

HTTP server is called from the `main()` function in an endless loop through `http_handler()` function [23]. Whenever the web browser sends a request to HTTP server, `http_handler()` is calling for an appropriate CGI function. CGI function is invoked with the only parameter – a pointer to `HttpState` structure. This structure contains necessary information for development of CGI functions. It is holding internal state variables of the HTTP server instance that is handling current request. Its content is valid only within a CGI function called from the HTTP server. Fields of `HttpState` structure that are used in the implementation of this project are:

- s – socket associated with the HTTP server. It allows TCP functions to be used such as `sock_read, sock_write,` etc.
- substate – this is used to hold current state of state machine for CGI function. Thus when a CGI function returns control back to the HTTP server, its values are preserved for the next time it is invoked. This field can be accessed through `http_getState()` and `http_setState()` functions and when CGI function is called for the first time it is set to 0.
- main_timeout – timeout for the server. Upon every call of `http_handler()` the web server checks against this timeout, and if it has been exceeded, current processing is terminated, and the server goes back to its initial state. The server resets this value when it changes states. This value is assigned with the help of the following macro:

  ```
  #define HTTP_TIMEOUT 16
  ```

  Thus 16 is the number of seconds until the web server will time out.

- buffer[] – a buffer to contain data to be transmitted over the socket. Its size is defaulted to 256 bytes. It is accessed by `http_getData()` and `http_getDataLength()` functions.

3.7.5.6 Upload functionality implementation

Unlike previous versions of Dynamic C when CGI functions were unable to handle large amount of data coming from the browser and thus were limited to processing simple forms, the latest version provides CGI functions with the enhanced capability of dealing with large data sets [23]. This is extremely important for the development of this project since one of its main purposes is to provide remote firmware upgrade for a reconfigurable FPGA system. Therefore this system has to be able to receive Mbytes of data.

In order to use this feature the following macro has to be defined:

```
#define USE_HTTP_UPLOAD
```

When HTTP server receives incoming data it separates parts and parses the headers. It then calls for the defined CGI function with data for each section and an appropriate action code. Current action code is determined by calling the `http_getAction()` function, which return value indicates the reason CGI was called by HTTP server. Possible action codes are:

- `CGI_START` – start of a new part of incoming data
- `CGI_DATA` – new chunk of data within current part
- `CGI_END` – end of current part of data, indicates the end of upload

- `CGI_EOF` – after the file has been uploaded, the action code is set to this state to tell the HTTP server to stop calling this CGI function.
- `CGI_ABORT` – upload has been terminated as a result of user pressing the cancel button or by the network.

Therefore, when a CGI function is invoked by `http_handler()` the first thing to do is to find out the reason it was called. `http_getAction()` returns the action code and afterwards each action code must be treated accordingly.

`CGI_START` – when the new data is incoming, HTTP server reads all the headers and thus possesses all the necessary information about the data. By calling `http_getContentLength(s)` function with `s` being a pointer to `HttpState` structure the length of the whole uploaded file can be found. Since the file is to be transmitted to the board containing the FPGA system under test, all necessary preparations for transmission are done at this stage.

`CGI_DATA` – at this point parts of uploaded file start coming and need to be processed. By calling `http_getData()` available data is retrieved and `http_getDataLength()` returns its length in bytes. It can be of any size up to the value defined in `HTTP_MAXBUFFER`, which was set to 256 bytes. The data is then transmitted to system under test.

`CGI_END` – this is to signal that the file has been uploaded in full. End of transmission indication is sent to system under test.

`CGI_EOF` – After the upload of the file has been completed, HTTP server has to go back to original web page and present it to the client. This is done by calling `http_switchCGI(s, "index.html")` with the second parameter specifying the location of that page.

`CGI_ABORT` – indicates loss of connection, therefore no point in handling anymore incoming data, HTTP server has to go to initial state.

3.7.6 Communication with System Under Test

Uploaded bit-stream file is being transmitted on the fly to the platform containing FPGA system and is saved in its internal memory. Upon the completion of receiving bit-stream file from the remote client, a notification is sent to the board and FPGA is reloaded with the new firmware.

Transmission of bit-stream file is done through serial port E, which is connected to RS-232 transceiver. Thus communication with the board is fulfilled through RS-232 interface.

3.7.6.1 Serial Port Setting

Serial port is set to the following parameters:

Baud Rate - 115200 bps

Data bits - 8 bits

Stop bits - One stop bit

Parity - None

Baud Rate is defined in the beginning:

```
#define BAUD232 115200
```

Also two buffers are defined for receiving data and transmission:

```
#define EINBUFSIZE  15

#define EOUTBUFSIZE 15
```

The rest of the parameters are configured upon opening the port E:

```
serEopen(BAUD232)
```

In order to work with 3-wire serial port instead of 5-wire, serial mode has to be configured:

```
serMode(0)
```

Upon the beginning of transmission both buffers for receiving and transmitting the data are emptied:

```
serEwrFlush()

serErdFlush()
```

3.7.6.2 Communication Protocol

To initiate the transmission, the following actions are required:

Transmit to platform: <0x41>

Receive from platform: <0x3E>

Before sending new bit-stream, all appropriate slots must be deleted. To achieve this, the following actions are required:

Transmit to platform: <0x41>, delay 100 ms

Transmit to platform: <0x65>, delay 100 ms

Transmit to platform: <0x72>, delay 100 ms

Transmit to platform: <0x73>, delay 100 ms

Transmit to platform: <0x6C>, delay 100 ms

Transmit to platform: <slot number> delay 40 seconds.

The board has several slot numbers for loading partial cores. Large delay of 40 seconds is needed due to the nature of the type of FLASH used.

Transmission of the bit-stream is performed as follows:

Transmit to platform: <0x6C >, delay 100 ms

Transmit to platform: <0x64 >, delay 100 ms

Transmit to platform: <0x63 >, delay 100 ms

Transmit to platform: <0x72 >, delay 100 ms

Transmit to platform: <0x01 >, delay 100 ms – this specifies the slot number

Transmit to platform: <0x00 >, delay 100 ms

Transmit to platform: <0x00 >, delay 100 ms

Transmit to platform: <0x00 >, delay 100 ms

Afterwards transmission of the file itself begins at full speed. Once the transmission is completed, a notification is sent to the platform:

Transmit <0xFF> 16 times without delays to request reset of FPGA.

Finally, the port is closed, and in 3 seconds the board will timeout and reload bit-stream located in slot #1.

3.8 Summary

In this chapter implementation concepts of e-Maintenance system for FPGA-based platform were presented. Its specification and architecture were determined. Detailed description of HTTP server implementation and communication with FPGA platform was described.

Chapter Four

Experimental Results

4.1 Introduction

Testing procedures and results for the prototype of e-Maintenance for FPGA-based reconfigurable system are described in this chapter. Its functionality verification and timing analysis are presented. Conclusions are drawn for other FPGAs based on the results.

4.2 Testing Methodology

In order to test the developed e-maintenance system several things are required. First a PC with web browser to function as a remote client. It should communicate via Internet with HTTP server on PowerCore Flex system. And finally, the platform containing the FPGA which is serially connected to PowerCore system. This scheme is presented in Figure 4.1.

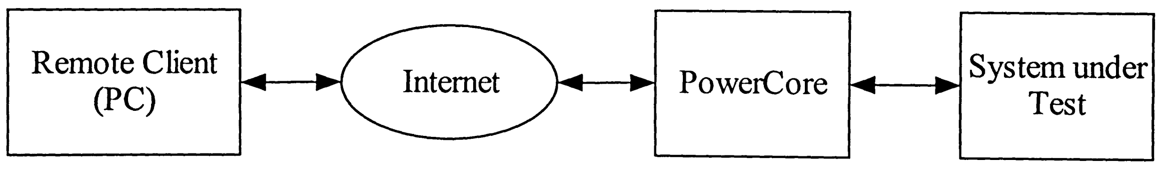


Figure 4.1. Testing Scheme.

4.3 Experimental Setup

The experimental setup for the prototype of e-Maintenance system for FPGA-based reconfigurable system is shown in Figure 4.2.

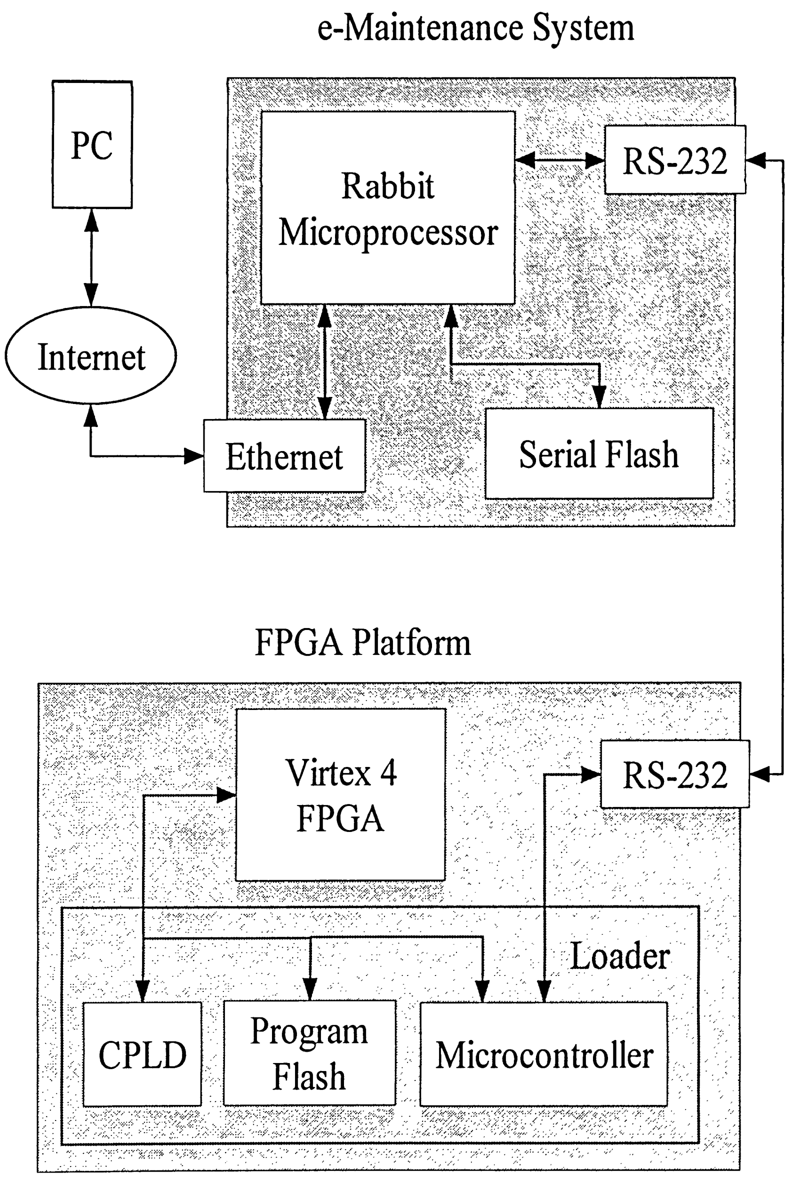


Figure 4.2. Experimental Setup for the prototype of e-Maintenance system.

Experimental setup consists of PowerCore FLEX board implementing the prototype of e-Maintenance system, Multi-Stream Adaptive Reconfigurable System (MARS), data interface to MARS, hub for Remote Client connection to PowerCore via Internet and HP54620C logic analyzer for system measurements. Photographic images of the setup are presented in Appendixes B1, B2 and B3 courtesy of ERSL.

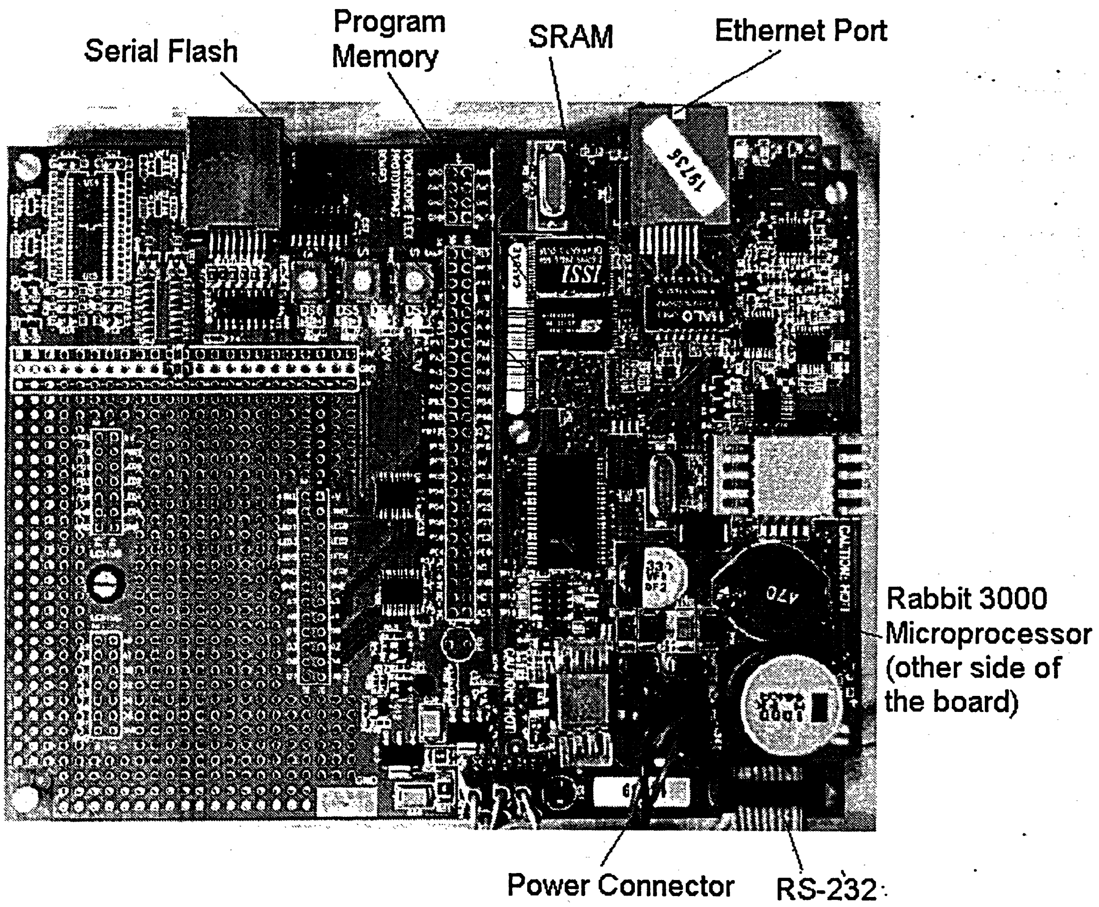


Figure 4.3. Photographic image of the e-Maintenance system.

Detailed photographic image of the e-Maintenance system is presented in Figure 4.3. It is implemented on the base of PowerCore FLEX module. It incorporates Rabbit 3000 Microprocessor which implements the HTTP server for the e-Maintenance system. Serial

Flash memory is used for HTML pages and forms storage and it can also provide storage for log files of system under test. Connection to Internet is attained via 10/100 Ethernet Port, and RS-232 header is used as an interface to FPGA platform.

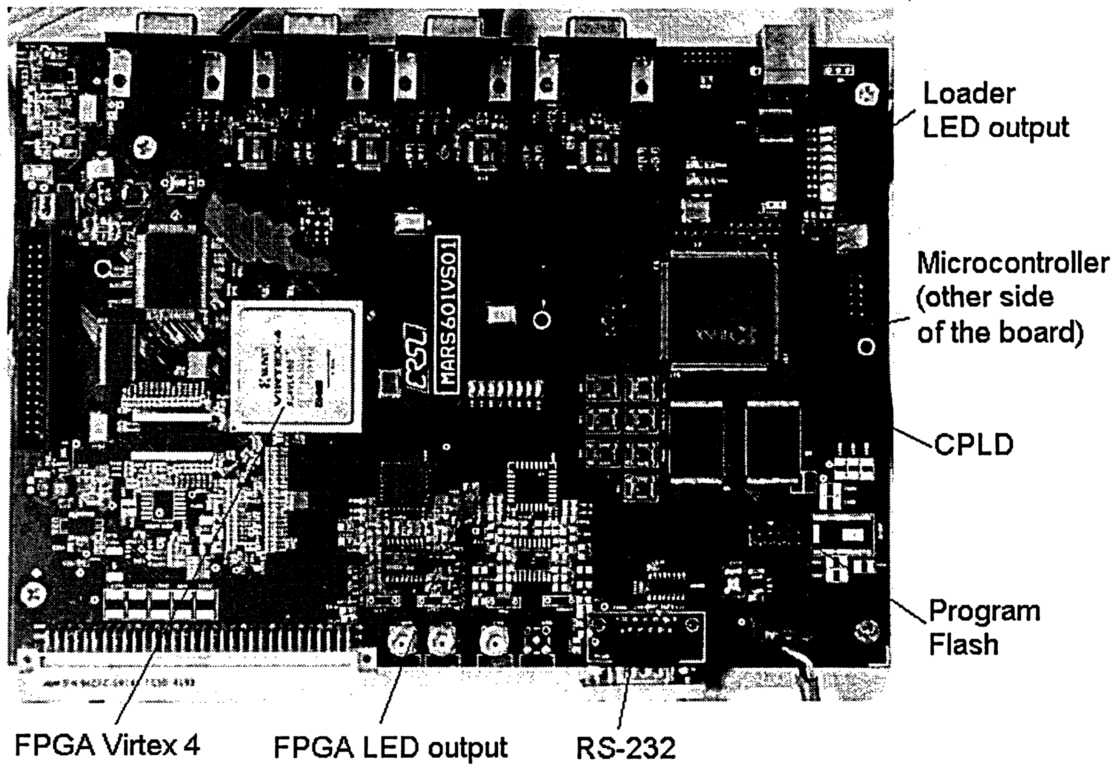


Figure 4.4. Detailed photographic image of MARS FPGA platform.

Picture courtesy of ERSL.

Detailed photographic image of MARS FPGA platform is shown in Figure 4.4. It incorporates the Virtex 4 XC4VLX160 FPGA. The loader consists of Microchip Pic18LF8410 microcontroller which receives the incoming bitstream through RS-232, CPLD XC95288XL for saving the bistream into Program Flash. CPLD is also used for loading the FPGA itself. Data connection between the loader and FPGA is SelectMAP32, which is a 32 bit data bus interface to the Virtex-4 configuration logic [25].

4.4 Setting the Network

In order to connect the network an Ethernet 10Base-T hub and two standard network cables are used. This way a Micro-LAN is set up. Both the development PC and PowerCore are connected to the hub using two straight through Ethernet cables, which in turn is connected to the Internet through the adapter. The whole setting is presented in Figure 4.5.

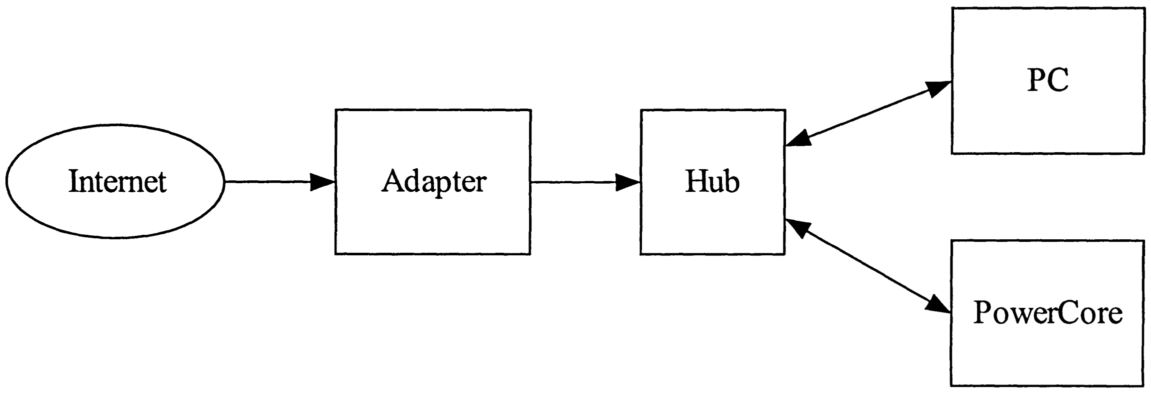


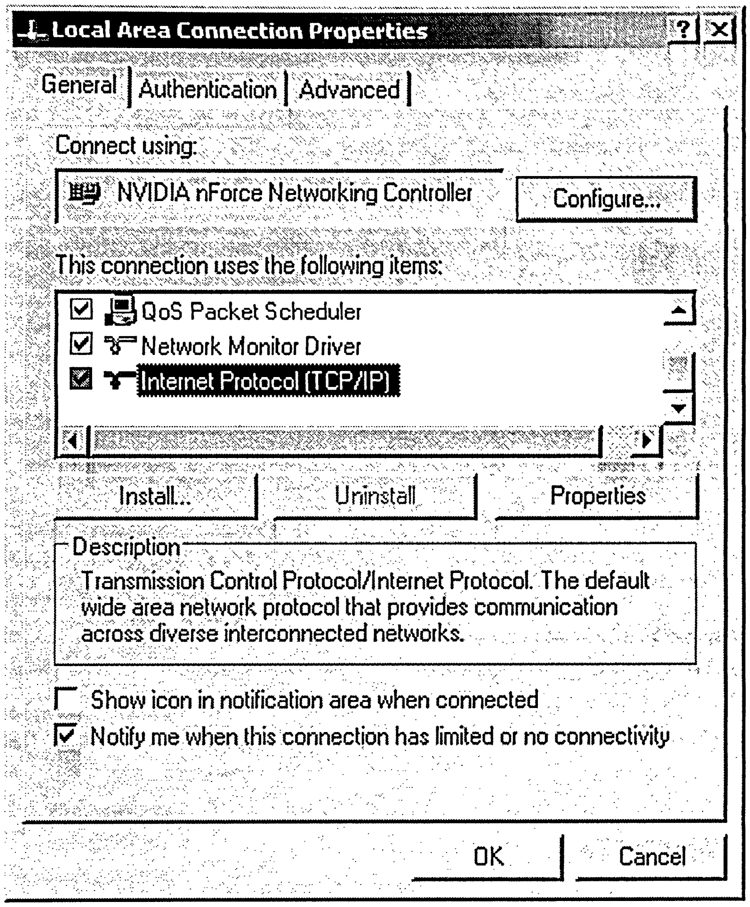
Figure 4.5. Network Setting.

4.5 Configuring the IP Address

The default configuration is used where the PowerCore module has its IP address set to 10.10.6.100 and the netmask is 255.255.255.0. The development PC is assigned the address 10.10.6.101 with the netmask 255.255.255.0. This is achieved by doing the following operations:

- Starting the Control Panel and selecting Network and Internet Connections
- Selecting Network Connections

- Right click on Local Area Connection and selecting the Properties
- In the Local Area Connection Properties window choosing Internet Protocol (TCP/IP) Properties as illustrated in Figure 4.6:


Local Area Connection Properties

General | Authentication | Advanced

Connect using:

NVIDIA nForce Networking Controller [Configure...]

This connection uses the following items:

☑ QoS Packet Scheduler
☑ Network Monitor Driver
☑ Internet Protocol (TCP/IP)

[Install...] [Uninstall] [Properties]

Description

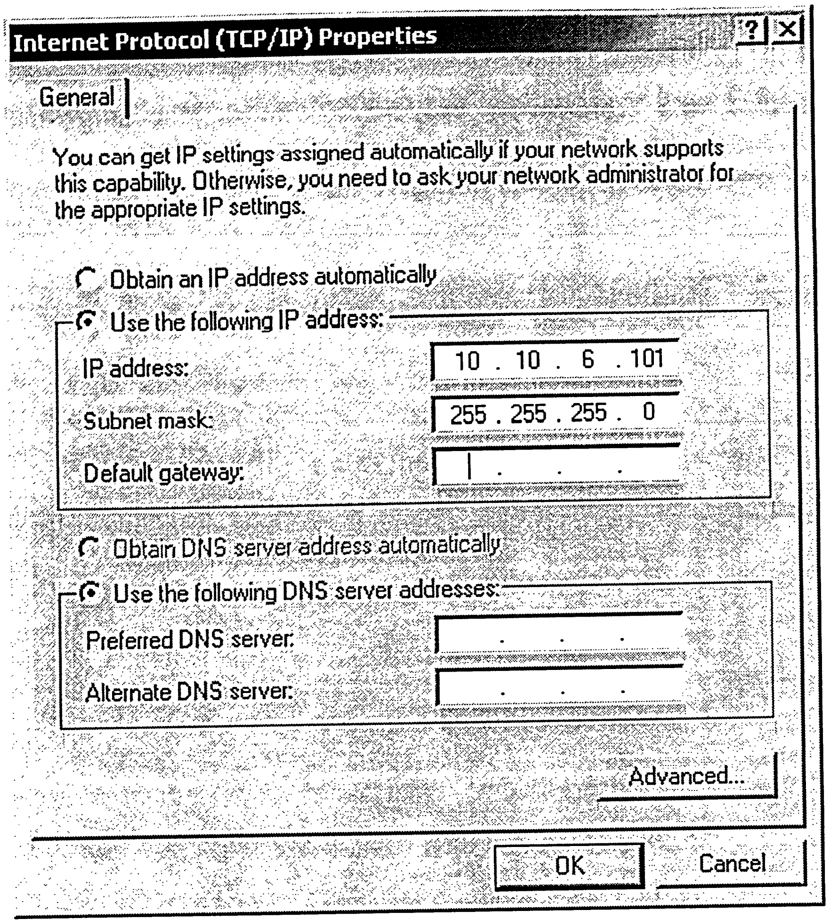
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

☐ Show icon in notification area when connected
☑ Notify me when this connection has limited or no connectivity

[OK] [Cancel]

Figure 4.6. Local Area Connection Properties.

- In the Internet Protocol (TCP/IP) Properties window click on Use the following IP address and enter IP address 10.10.6.101 and netmask 255.255.255.0. Figure 4.7. illustrates that:


Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 10 . 10 . 6 . 101

Subnet mask: 255 . 255 . 255 . 0

Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

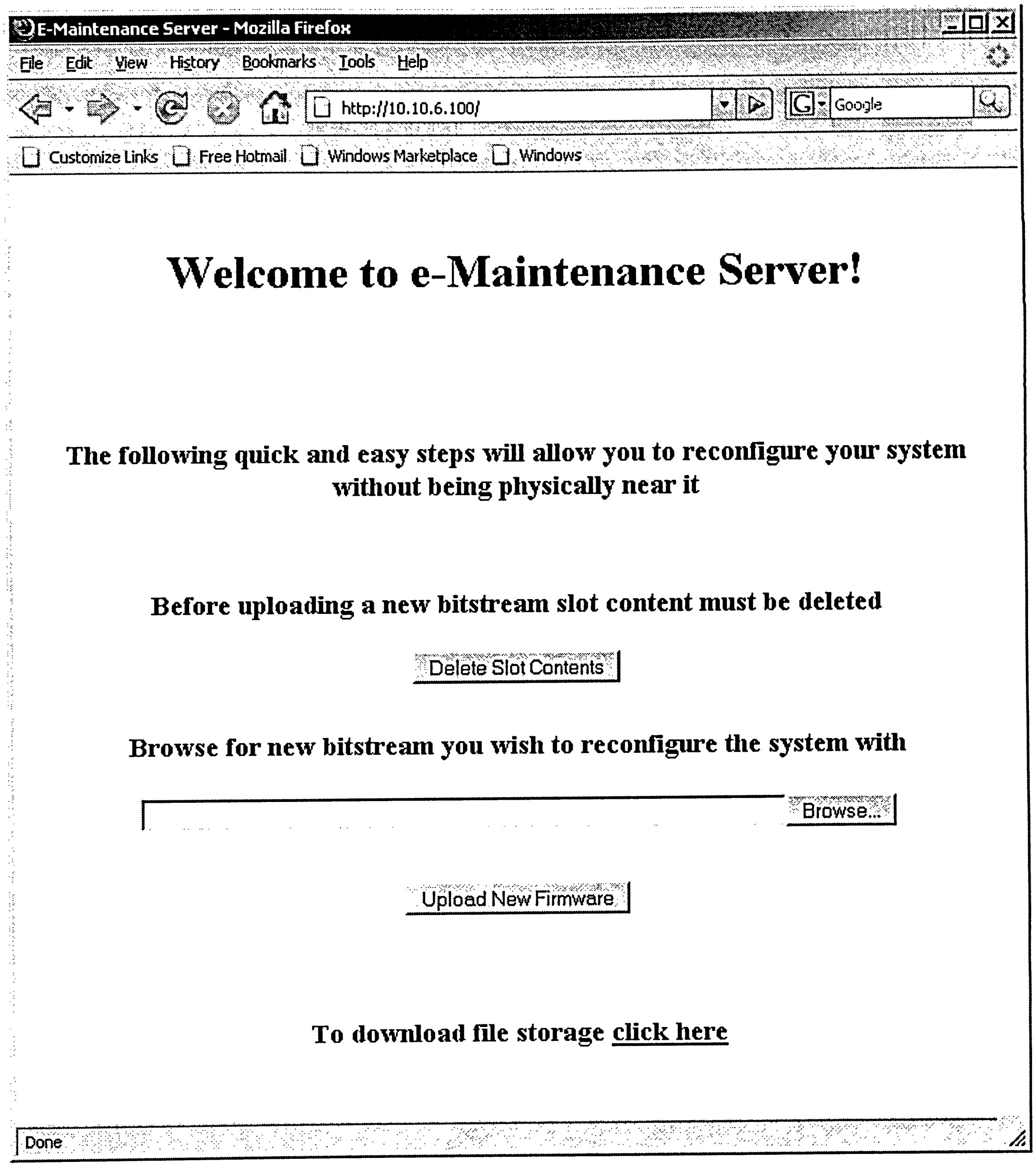
Preferred DNS server: . . .

Alternate DNS server: . . .

Figure 4.7. Setting Internet Protocol Properties.

4.6 HTTP Server Test

After the network has been set up, HTTP server is ready for testing. A client can connect to it and send requests through web browser. Upon receiving HTTP packets the server will be sending web pages back to the browser. When the server IP address 10.10.6.100 is entered in web browser's address bar, home page of the server is displayed. Figure 4.8 shows the snapshot of that page:


E-Maintenance Server - Mozilla Firefox

Welcome to e-Maintenance Server!

The following quick and easy steps will allow you to reconfigure your system without being physically near it

Before uploading a new bitstream slot content must be deleted

Browse for new bitstream you wish to reconfigure the system with

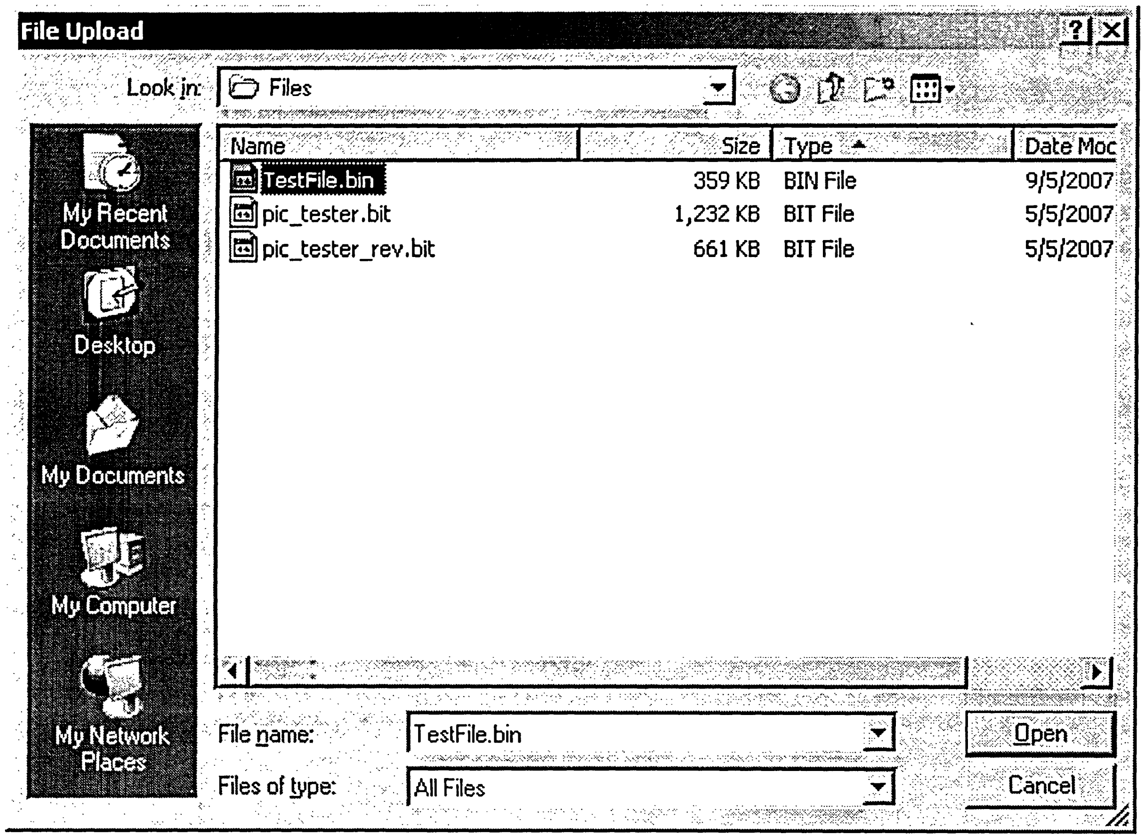
To download file storage click here

Figure 4.8. Home Page of e-Maintenance Server.

When homepage appears in the browser it means that network connection was established properly and HTTP server is functioning.

4.7 Upload and Download Functionality Test

In order to test the upload functionality a test file has to be chosen through the browser. By pressing the Browse button the dialog presented in Figure 4.9 is opened.

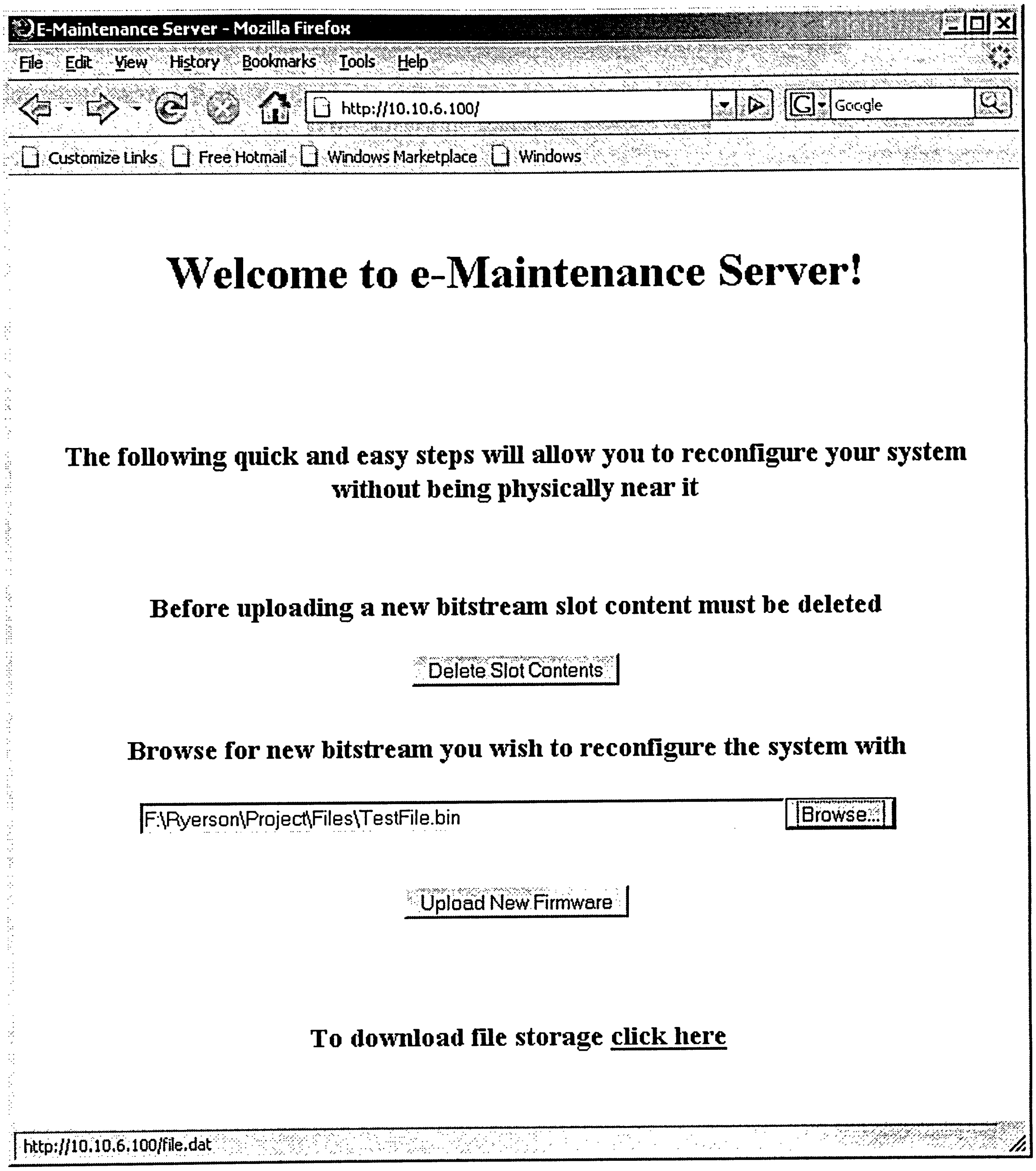


Name	Size	Type	Date Moc
TestFile.bin	359 KB	BIN File	9/5/2007
pic_tester.bit	1,232 KB	BIT File	5/5/2007
pic_tester_rev.bit	661 KB	BIT File	5/5/2007

File name: TestFile.bin
Files of type: All Files

Figure 4.9. File Browse Dialog from HTTP Server.

For this test TestFile.bin with size 359 KB will be used. After the file is chosen, it appears in the edit box of the main page as shown in Figure 4.10. After pressing the Upload New Firmware button, the file is sent to HTTP server. As the server receives data packets it saves the file in the serial Flash for download functionality test to be conducted right afterwards.


E-Maintenance Server - Mozilla Firefox

Welcome to e-Maintenance Server!

The following quick and easy steps will allow you to reconfigure your system without being physically near it

Before uploading a new bitstream slot content must be deleted

Browse for new bitstream you wish to reconfigure the system with

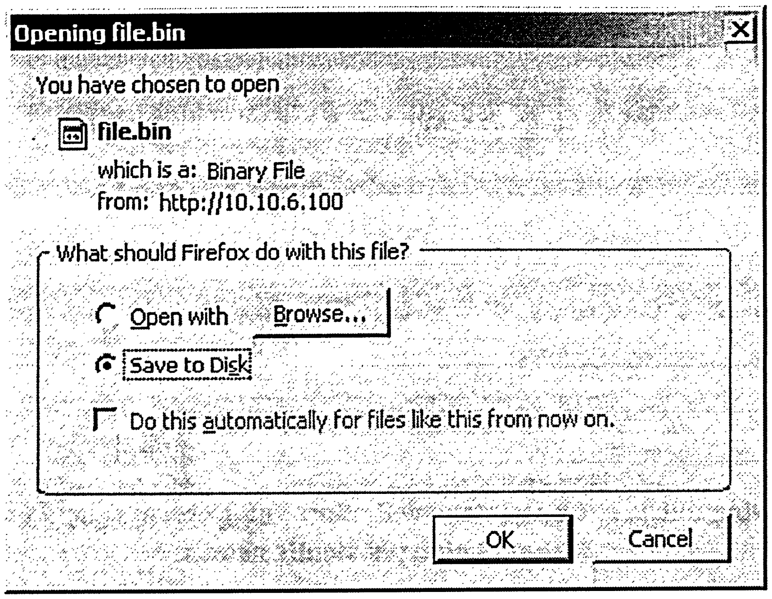
F:\Ryerson\Project\Files\TestFile.bin

To download file storage click here

http://10.10.6.100/file.dat

Figure 4.10. Selected test file.

Once the upload of the test file has been completed, the download functionality can be tested. By clicking the download file storage option on the bottom of the web page the dialog presented in Figure 4.11 is invoked.


Opening file.bin

You have chosen to open

file.bin

which is a: Binary File

from: http://10.10.6.100

What should Firefox do with this file?

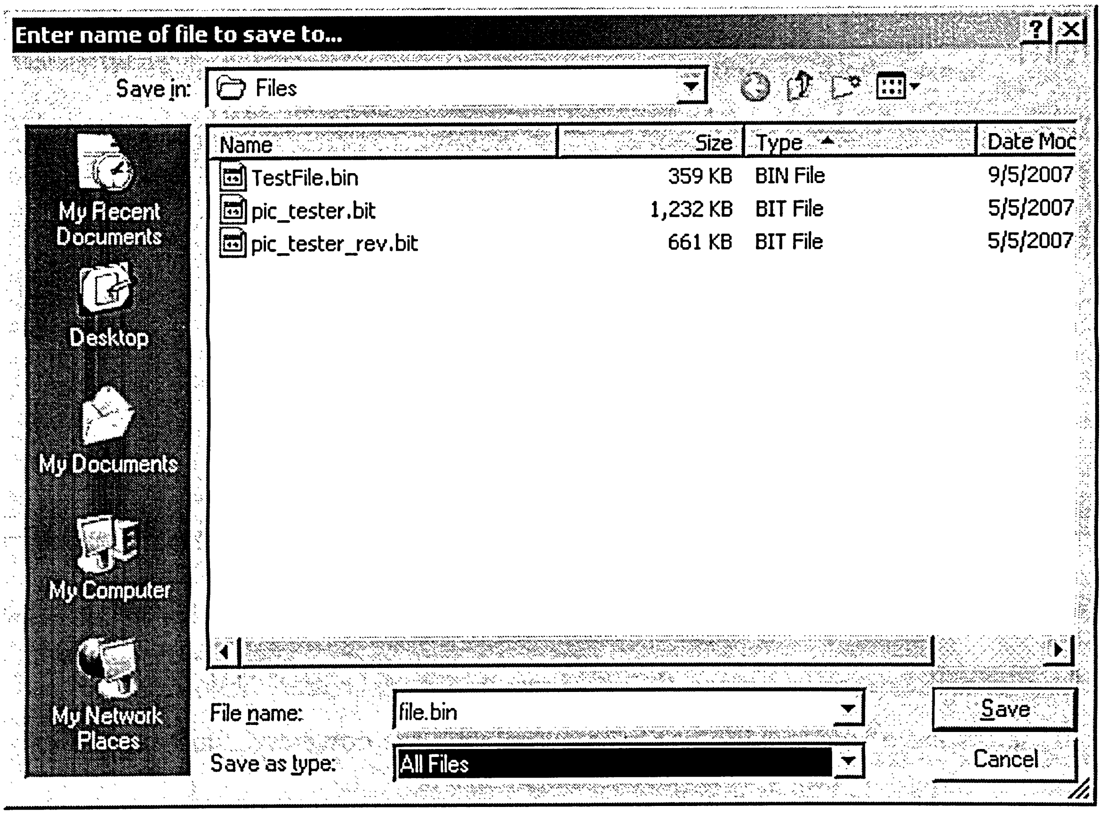
( ) Open with Browse...

[✓ Save to Disk]

☐ Do this automatically for files like this from now on.

OK Cancel

Figure 4.11. File Download Dialog.


Enter name of file to save to...

Save in: Files

Name	Size	Type	Date Moc
TestFile.bin	359 KB	BIN File	9/5/2007
pic_tester.bit	1,232 KB	BIT File	5/5/2007
pic_tester_rev.bit	661 KB	BIT File	5/5/2007

File name: file.bin

Save as type: All Files

Figure 4.12. Saving File Dialog.

The file is to be saved under different name so that it can be compared to the original later as shown in Figure 4.12. Once the file is downloaded its contents and size is compared to those of the original file. They are exactly the same, therefore the upload and download functionalities test is successful.

4.8 Communication with FPGA Platform Test

E-Maintenance system is connected to FPGA platform with a serial cable with RS-232 connectors. For communication between two platforms special communication protocol has been developed as was discussed in previous chapter. FPGA-based system has several slots for reconfiguring specific partitions. Before uploading new firmware slot contents must be deleted. In order to do that on the web browser Delete Slot Contents button must be pressed. E-Maintenance system sends appropriate commands to FPGA-based system in accordance to communication protocol and once the slot contents have been deleted the platform responds by LED flashing. This means that communication test between two platforms has passed successfully.

4.9 Remote FPGA Reconfiguration Test

Once slot contents have been deleted, new firmware can be uploaded to the system. The bit stream is selected through the web browser as was shown in section 4.7. Once Upload New Firmware button is pressed, selected bit stream is uploaded to e-Maintenance server via Internet and is transmitted to FPGA platform through serial communication. Upon completion of the upload, appropriate protocol commands are sent to the platform in order to reset the FPGA and to reconfigure it with new firmware. Once the reconfiguration is complete, FPGA LED output signals that the reconfiguration was successful.

4.10 Timing Analysis of Remote Reconfiguration of the FPGA

Measurements of the system were conducted using HP54620C logic analyzer, see Figure 4.13. The results were attained by measuring the RS-232 TX and RX signals on the PowerCore FLEX module.

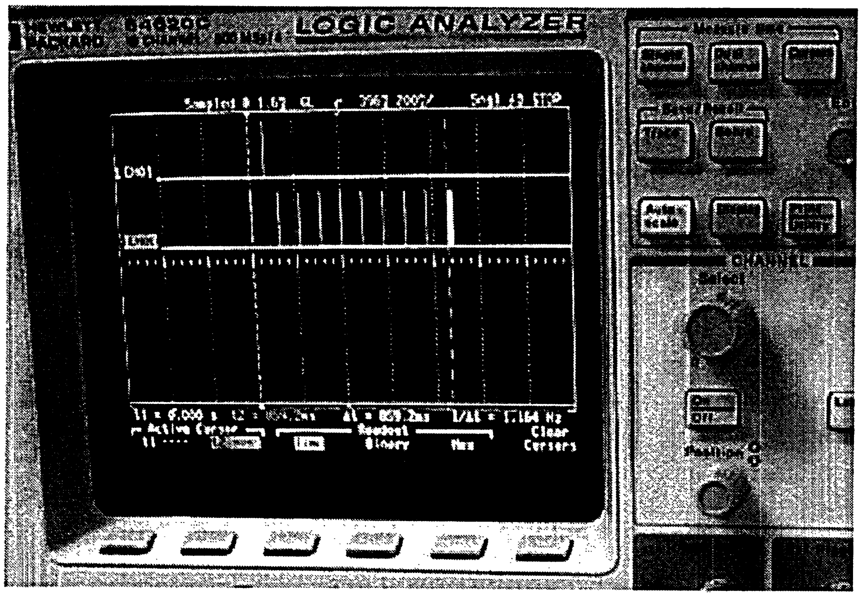


Figure 4.13. HP54620C logic analyzer used as a measuring tool for the system.

Picture courtesy of ERSL.

4.10.1 Baud Rate Test

As was discussed in chapter 3 according to communication protocol between the e-Maintenance system prototype and FPGA-based platform the baud rate was set to 115,200 bps with 8 data bits and one stop bit. Figure 4.14 examines the correctness of the baud rate. It shows that 9 bits were transmitted in 78.4 μs. Therefore one bit is transmitted in 8.71 μs whereas in theory it should be 8.68 μs which is a very close result.

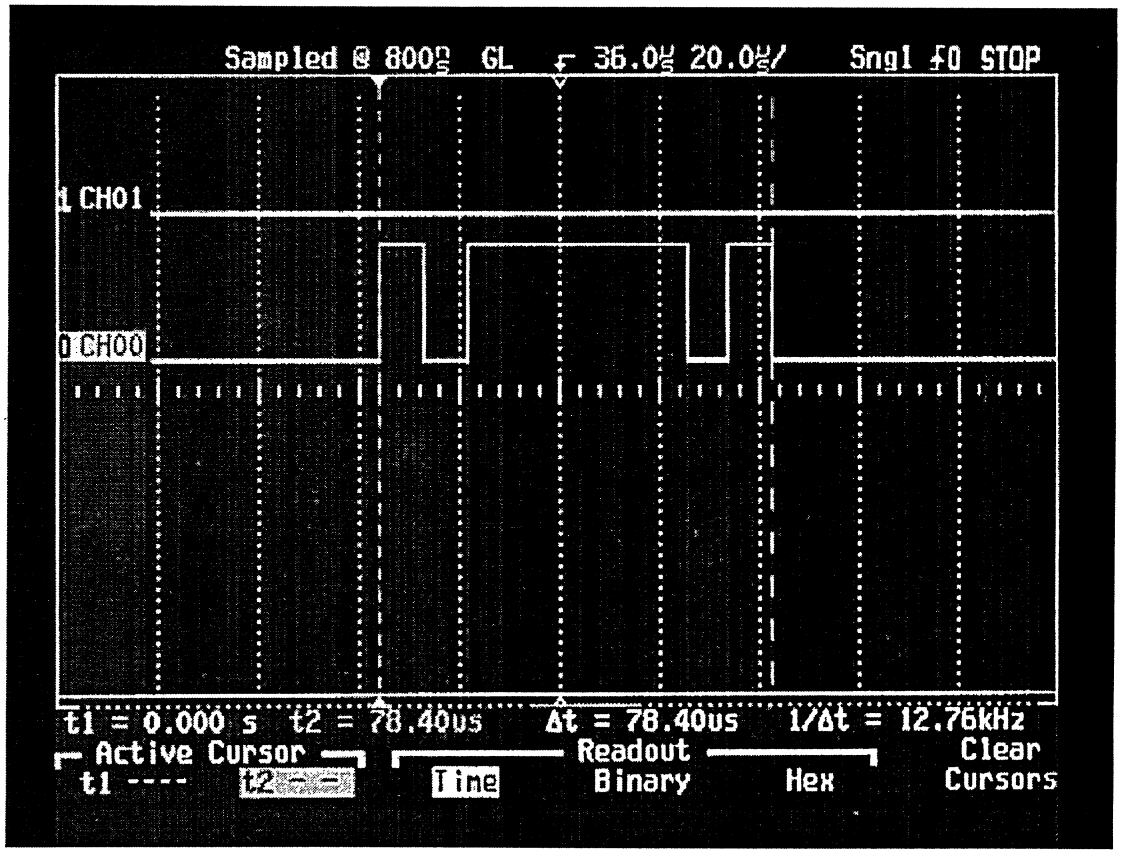

Sampled @ 800ns GL 36.0μs 20.0μs/ Sngl ⨍0 STOP
1 CH01
0 CH00
t1 = 0.000 s t2 = 78.40us Δt = 78.40us 1/Δt = 12.76kHz
Active Cursor
t1 ---- t2 ----
Readout
Time
Binary
Hex
Clear Cursors

Figure 4.14. Baud Rate Test. Picture courtesy of ERSL

4.10.2 FPGA Platform Response Time

In order to initiate transmission to FPGA platform e-Maintenance system has to transmit <0x41> according to communication protocol established between two platforms. In turn the FPGA platform is expected to respond with <0x3E> within 100ms. FPGA platform response time measurement is 71.2 ms as presented in Figure 4.15.

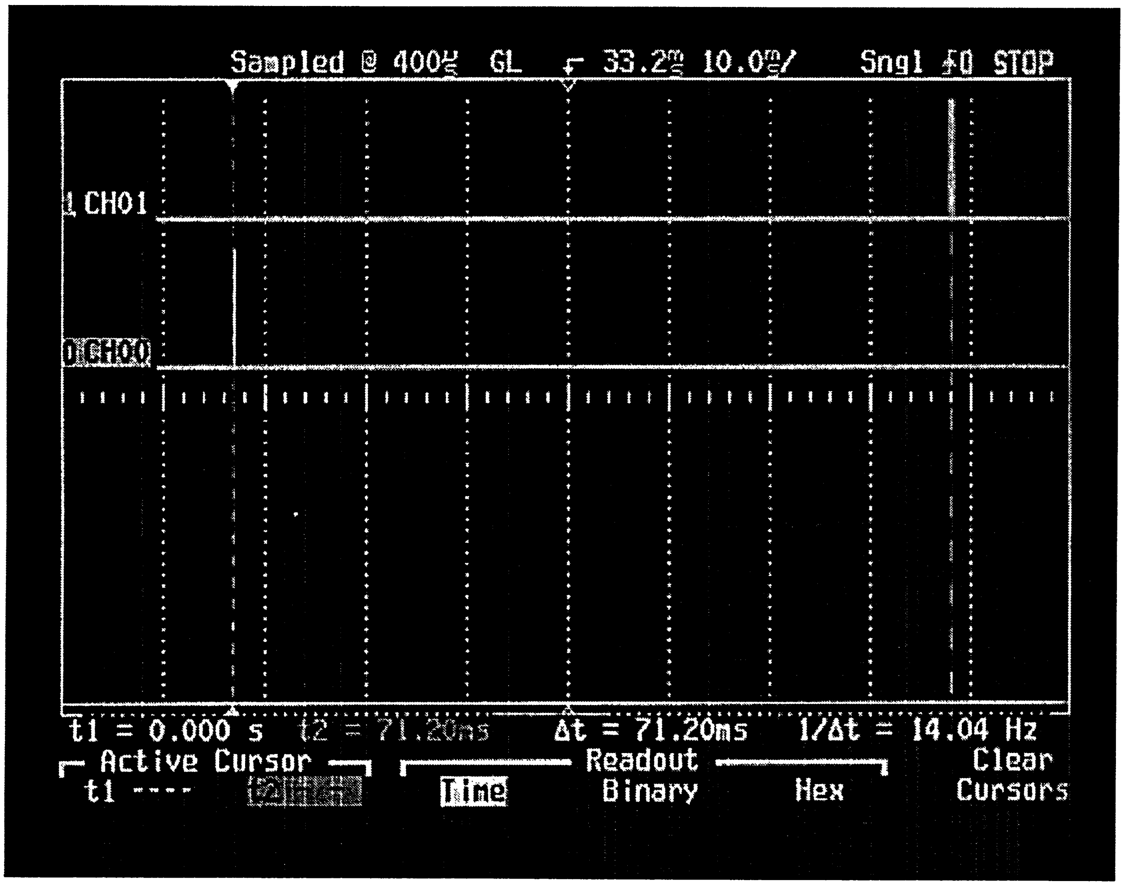
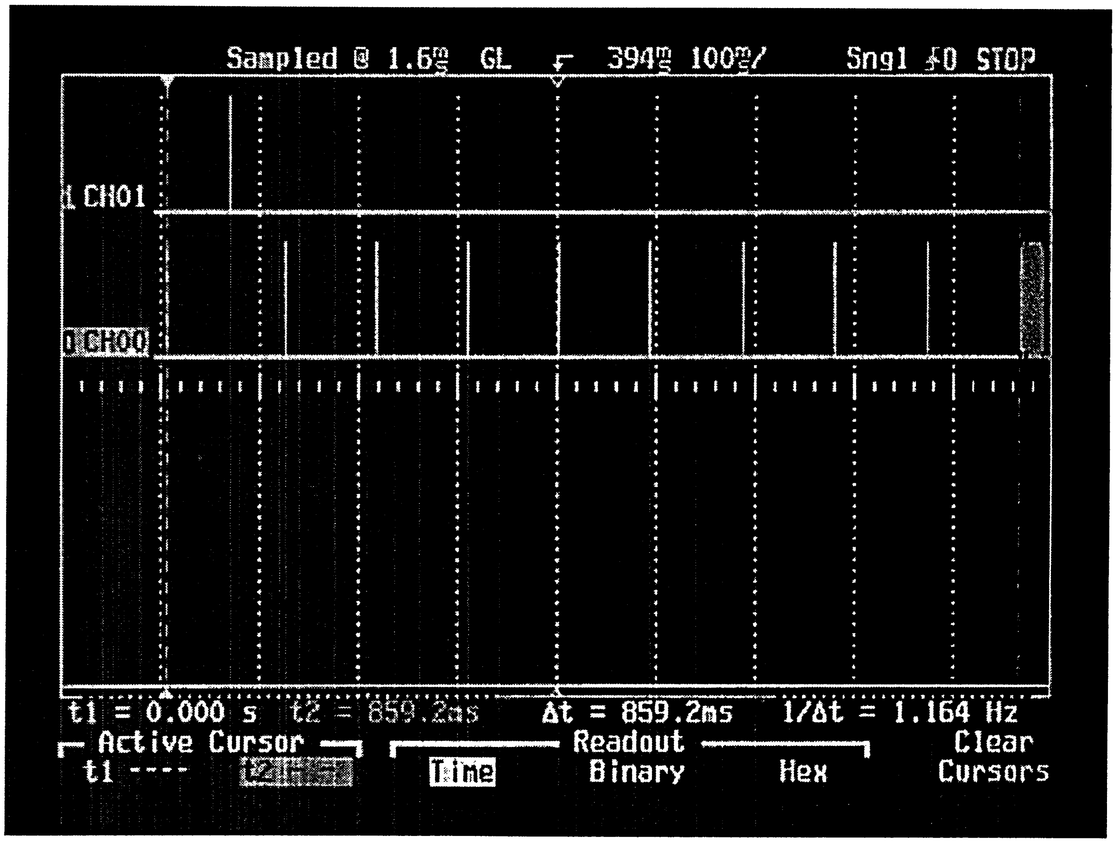

Sampled @ 400 GL 33.2 10.0 Sngl STOP
CH01
CH00
t1 = 0.000 s t2 = 71.20ms Δt = 71.20ms 1/Δt = 14.04 Hz
Active Cursor
Readout
Clear
t1
Time
Binary
Hex
Cursors

Figure 4.15. FPGA platform response time. Picture courtesy of ERSL.

4.10.3 File Transmission Initialization Time

File transmission initialization time result is presented in Figure 4.16. After the transmission initialization command from e-Maintenance system was sent and response was received from FPGA platform appropriate transmission commands are sent followed by first data packet. Time elapse from the transmission initialization to first data packet was 859.2 ms.



Sampled @ 1.6ms GL 394ms 100ms/ Sngl 0 STOP
1 CH01
0 CH00
t1 = 0.000 s t2 = 859.2ms Δt = 859.2ms 1/Δt = 1.164 Hz
Active Cursor
t1 ---- t2
Readout
Time
Binary
Hex
Clear Cursors

Figure 4.16. File Transmission Initialization Time. Picture courtesy of ERSL

4.10.4 Bitstream Upload Time

First uploaded bitstream was of 746,055 bytes in size and its upload time was 3 min and 55 sec as measured by a counter implemented inside e-Maintenance system. Second uploaded bistream was of 5,043,452 bytes in size and it was uploaded in 26 min and 10 sec.

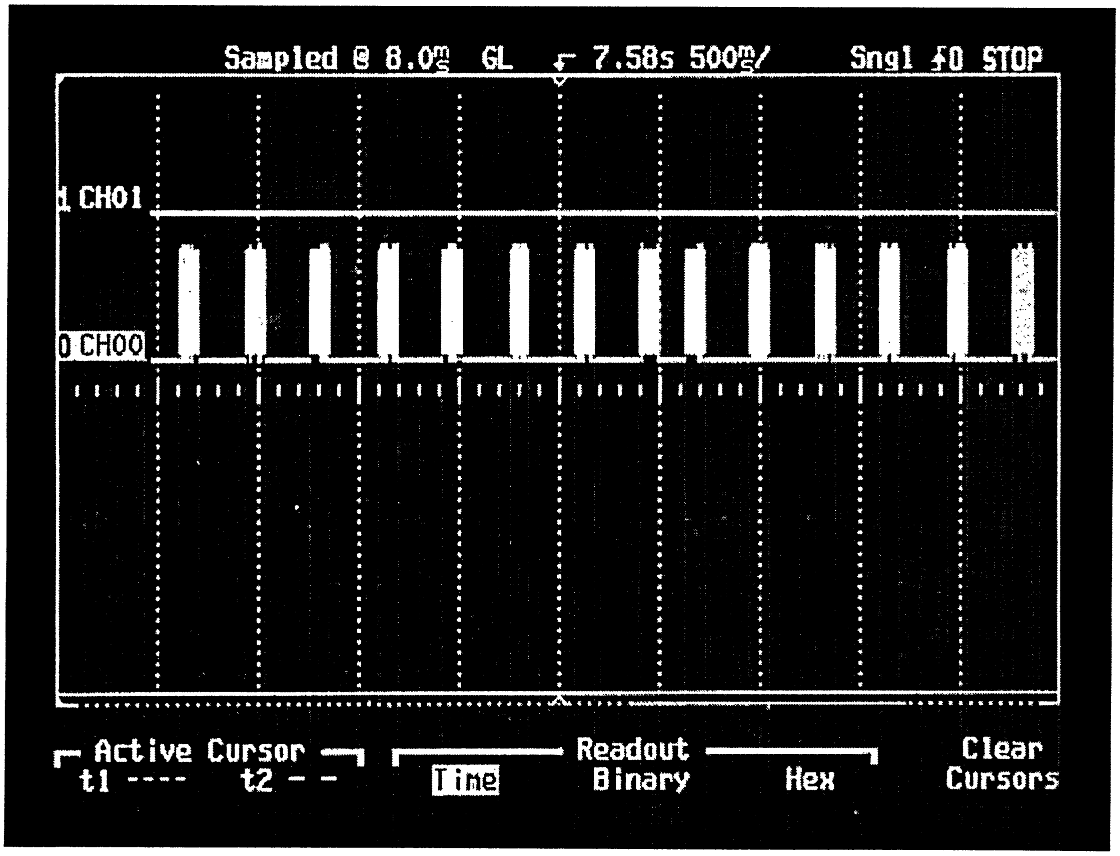


Figure 4.17. Transmitted Bitstream Packets. Picture courtesy of ERSL

Based on these measurements variation error for bitstream uploaded time can be calculated: file size ratio is 5,043,452 bytes/746,055 bytes = 6.76. Therefore the expected upload time of the second file is upload time of the first file multiplied by that ratio that is

235 sec x 6.76 = 1588.6 sec or 26 min and 28 sec. Thus the variation error is 1.1%. This is due to the fact that uploading of the file is done via Internet and transmission of data packets is asynchronous. Figure 4.17 presents this issue, it can be observed that time periods between data packets varies between 100 ms to 250 ms.

4.11 Estimation of Remote Reconfiguration for Xilinx Virtex 4 Family of FPGAs

Based on the timing analysis above the analytical model for estimation of remote reconfiguration for Xilinx Virtex 4 family of FPGAs can be created. The formula for reconfiguration time will be as follows:

Total Time = Transmission Initialization Time + Bistream Upload Time + Loader Switch Time + FPGA Reconfiguration Time.

Loader switch is done automatically as part of remote FPGA reconfiguration process. After the bitstream file is fully uploaded, e-Maintenance system sends appropriate sequence of commands and FPGA is reset. Depending on the size of FPGA reset time varies between 300 us to 500 us.

FPGA reconfiguration time is calculated according to the following formula:

Time = (File size – Header size) / (Uploaded bytes x Uploading Frequency).

Header size is 72 bytes. 4 bytes are uploaded at a time since FPGA reconfiguration data bus is 32 bits. Uploading frequency for which the measurements have been conducted is 50 MHz.

Estimated remote reconfiguration time for Virtex-4 family of FPGAs based on their configuration size in Xilinx Virtex-4 Product Table [26] and the analytical model presented above is summarized in Table 4.1.

Table 4.1. Estimated remote reconfiguration time for Virtex-4 family of FPGAs

FPGA	Configuration Memory Bits	Remote Reconfiguration Time
XC4VLX15	4,765,568	3 min 8.05 sec
XC4VLX25	7,819,904	5 min 22.9 sec
XC4VLX40	12,259,712	8 min 8.45 sec
XC4VLX60	17,717,632	11 min 38.49 sec
XC4VLX80	23,291,008	15 min 17.9 sec
XC4VLX100	30,711,680	20 min 10.15 sec
XC4VLX160	40,347,008	26 min 10.9 sec
XC4VLX200	51,367,808	33 min 43.42 sec
XC4VSX25	9,147,648	6 min 1.06 sec
XC4VSX35	13,700,288	9 min 0.31 sec
XC4VSX55	22,745,216	14 min 56.44 sec
XC4VFX12	4,765,568	3 min 8.52 sec
XC4VFX20	7,242,624	4 min 46.05 sec
XC4VFX40	13,550,720	8 min 54.42 sec
XC4VFX60	21,002,880	13 min 47.84 sec
XC4VFX100	33,065,408	21 min 42.79 sec
XC4VFX140	47,856,896	31 min 25.18 sec

4.12 Summary

In this chapter e-Maintenance system prototype was tested. Testing setup was configured, e-Maintenance system functionality was verified, its timing analysis was presented and estimations for Virtex-4 Family of FPGAs were calculated based on the results.

Chapter Five

Conclusions

In this project an embedded e-Maintenance for an FPGA-based reconfigurable system was developed. First, a study on system maintenance in general was conducted, followed by overview of Internet communication and study of main Internet Protocols, such as HTTP and TCP/IP. FPGA-based systems were described, their advantages and utilizations were discussed.

E-Maintenance system development was presented. HTTP server was implemented for remote connection with the system with upload functionality to receive new firmware over the internet and download functionality to make possible for service personnel to view log files of the system and conduct system diagnosis. Remote dynamic reconfiguration of FPGA-based system was successfully achieved. Partial reconfiguration of a chosen partition was also demonstrated. Timing analysis of FPGA-based system reconfiguration was conducted. An estimation of a whole family of FPGA reconfiguration was presented.

Successful implementation of this project opens possibilities for further study and development of e-Maintenance for reconfigurable systems. Future development work may include increasing upload speed by utilizing parallel communication between e-Maintenance system board and FPGA platform, developing tests for e-diagnosis of

FPGA that would enable locating faulty partitions and automatic reloading with a corresponding bit stream, testing the accuracy of data received over the internet, adding internet security and developing an e-Maintenance system servicing multiple FPGA-platforms simultaneously.

References

[1] V. Kirischian, V. Geurkov, P. W. Chun and L. Kirischian, "Reconfigurable Macro-Processor - Cost-Efficient Platform for Rapid Prototyping", in Proc. of 17-th International Conference FAIM-2007, Philadelphia, USA, June 2007, pp. 781-788.

[2] M.L. Bushnell, V.D. Agrawal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits", *Kluwer Academic Publishers*, 2000.

[3] V. Izosimov, P. Pop, P. Eles, Z. Peng, "Mapping of Fault-Tolerant Applications with Transparency on Distributed Embedded Systems", *9th EUROMICRO Conference on Digital System Design: Architecture, Methods and Tools*, pp. 313-322, 2006.

[4] M. Abramovici, M. Breuer, A. Friedman "Digital Systems Testing and Testable Design", *Wiley-IEEE Computer Society Press*, 1994.

[5] R. Itschner, C. Pommerell, M. Rutishauser, "Remote Monitoring of Embedded Systems in Power Engineering", *IEEE Internet Computing*, vol. 02, no. 3, pp. 46-52, 1998.

[6] S. Deb, S. Ghoshal, V.N. Malepati, D.L. Kleinman, "Tele-diagnosis: remote monitoring of large-scale systems", *Proc. IEEE Conf. Aerospace*, vol. 6, pp. 31-42, 2000.

[7] G. Walters, E. King, R. Kessinger, R. Fryer, "Processor Design and Implementation for Real-Time Testing of Embedded Systems", *Proc. IEEE Conf. Digital Avionics Systems*, vol. 1, pp. B44/1 – B44/8, 1998.

[8] M. Hung F. Cheng S. Yeh, “Development of a Web-services-based e-diagnostics framework”, *Proc. IEEE Int’l Conf. Robotics and Automation*, vol. 1, pp. 596-603, 2003.

[9] G. Held, "The ABCs of TCP/IP", *Aerbach Publications*, 2003.

[10] P. Loshin," TCP/IP clearly explained", *Academic*, Third edition, 1999.

[11] C. Maxfield, "The design Warrior's Guide to FPGAs", *Newnes Publishers*, 2004.

[12] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, P. Graham, “The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets”, *11th IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 133-142, 2003.

[13] M. Violante, M. Ceschia, M. Sonza Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, "Analyzing SEU Effects in SRAM-based FPGAs", *Proceedings of the 9th IEEE International On-Line Testing Symposium*, pp. 119-123, 2003.

[14] W. Quddus, A. Jas, N.A. Touba, “Configuration self-test in FPGA-based reconfigurable systems”, *Proc. IEEE Int’l Symp. Circuits and Systems*, vol. 1, pp. 97-100, 1999.

[15] N. Dorairaj, E. Shiflet, M. Goosman, “PlanAhead Software as a Platform for Partial Reconfiguration”, *XCell Journal*, pp. 68-71, 2005.

[16] D. Eisenreich, B. DeMuth, “Designing Embedded Internet Devices”, *Newnes Publishers*, 2003.

[17] Atmel - Data Sheet for AT90S8515, Atmel corporation, www.atmel.com.

[18] Microchip - PIC16F877 Data Sheet, Microchip Technology Inc., www.microchip.com.

[19] Microchip - Application Notes, Microchip Technology Inc., www.microchip.com.

[20] PowerCore FLEX Data Sheet, Z-World Inc., www.zworld.com.

[21] Rabbit 3000 Microprocessor User's Manual, Z-World Inc., www.zworld.com.

[22] Dynamic C User's Manual, Z-World Inc., 2005, www.zworld.com.

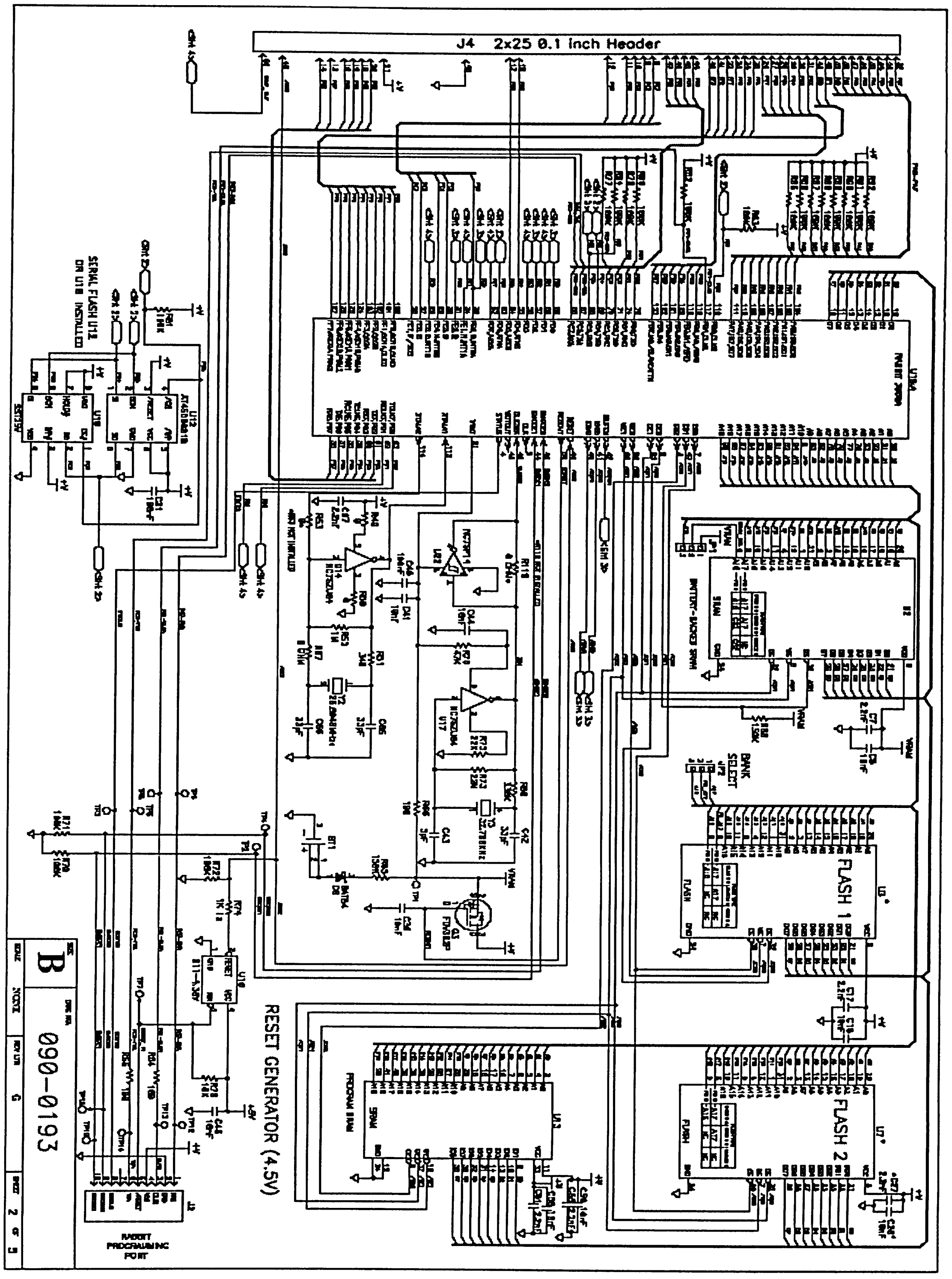
[23] Dynamic C TCP/IP User's Manual, Z-World Inc., 2005, www.zworld.com.

[24] PowerCore User's Manual, Z-World Inc., www.zworld.com.

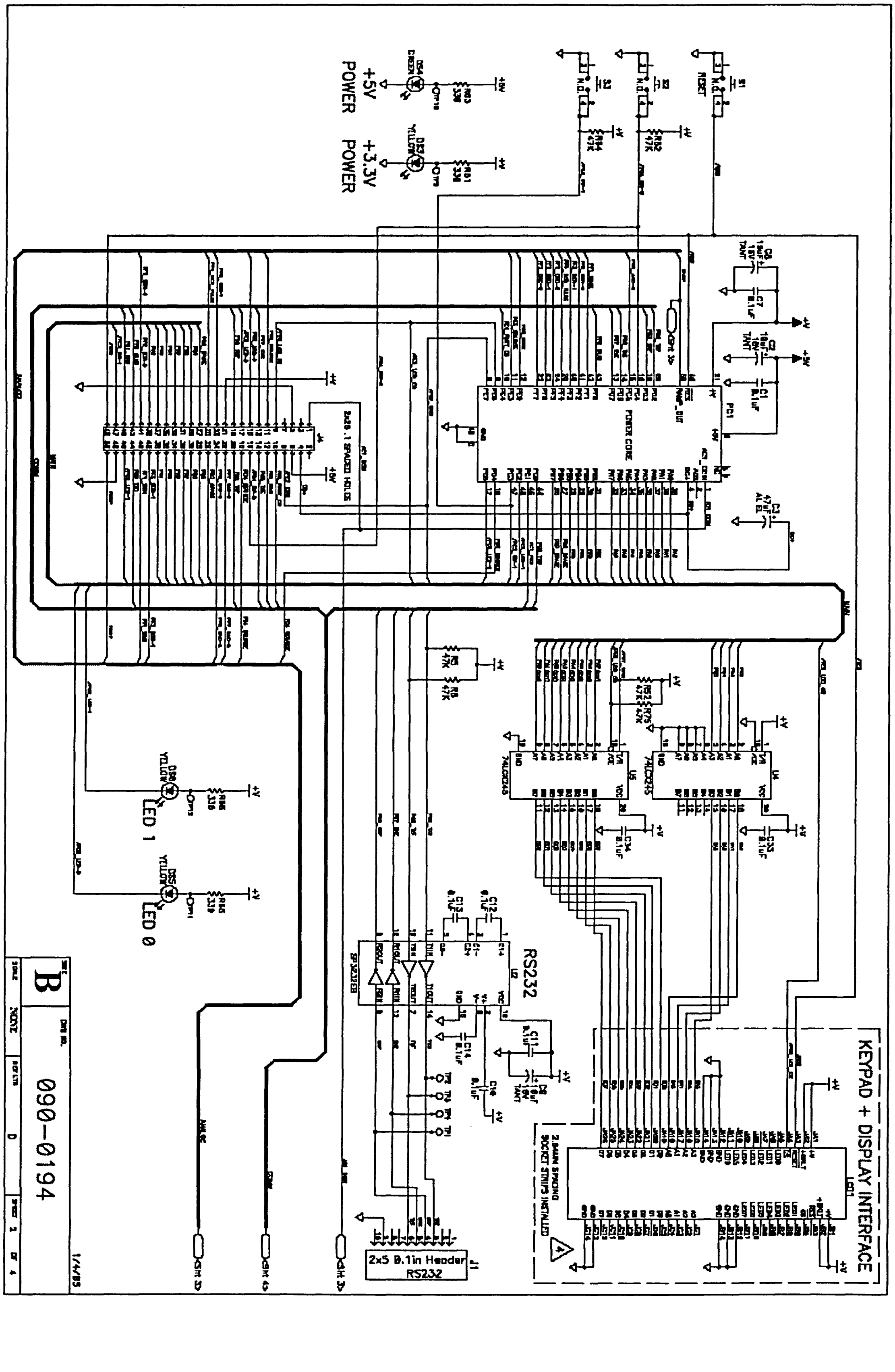
[25] Xilinx Virtex-4 Configuration Guide, www.xilinx.com.

[26] Xilinx Virtex-4 Product Table, www.xilinx.com.

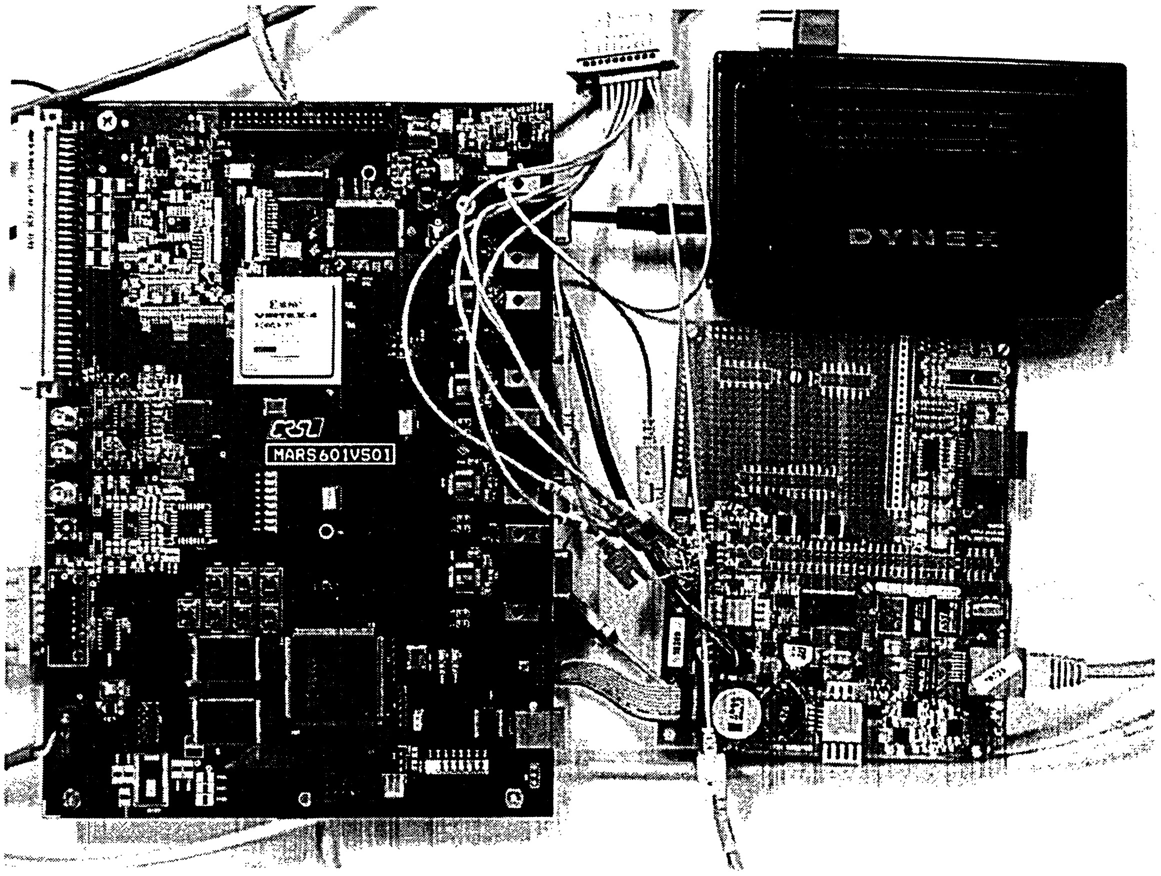
Appendix A1 - Schematic Diagram PowerCore FLEX



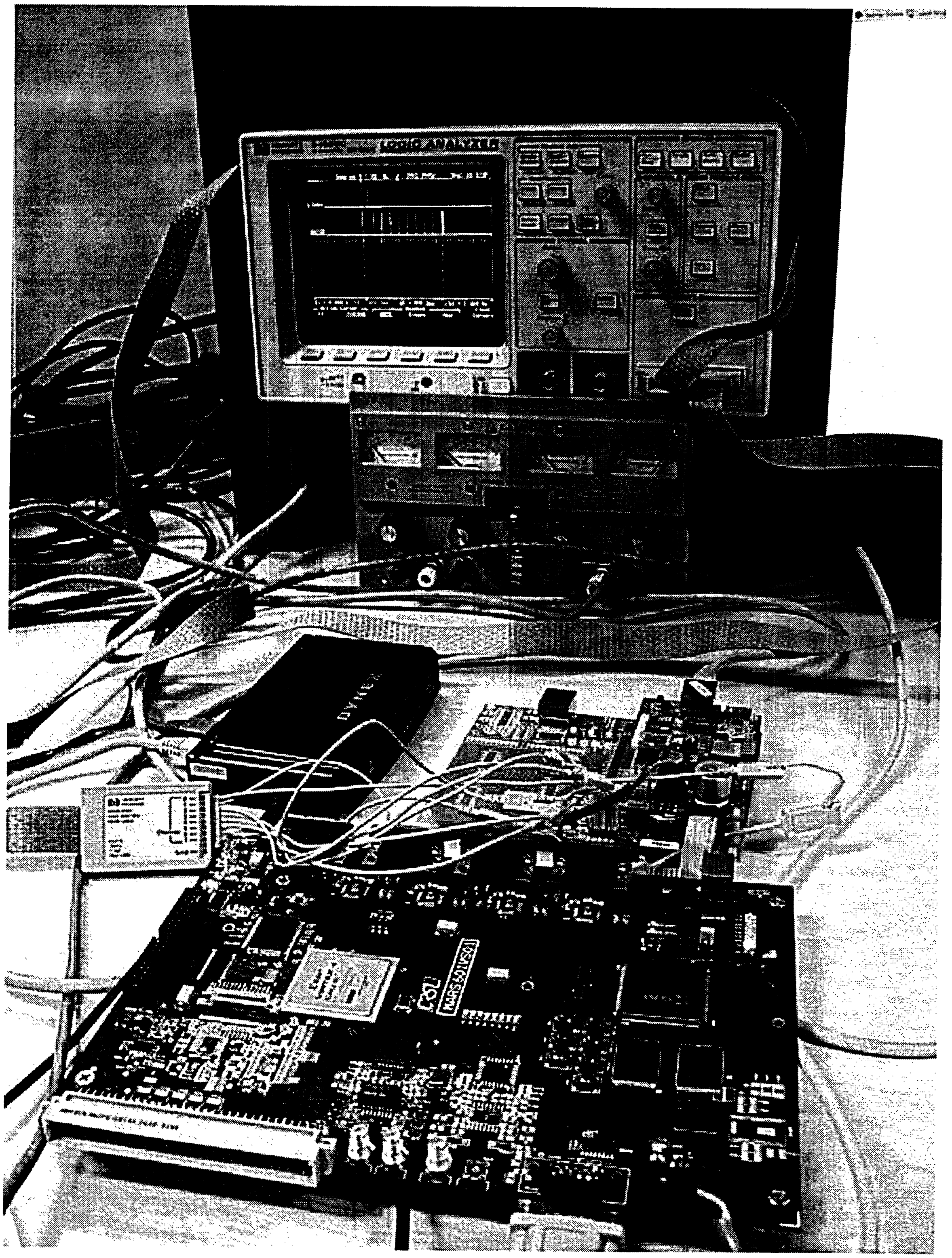
Appendix A2 - Schematic Diagram Prototyping Board



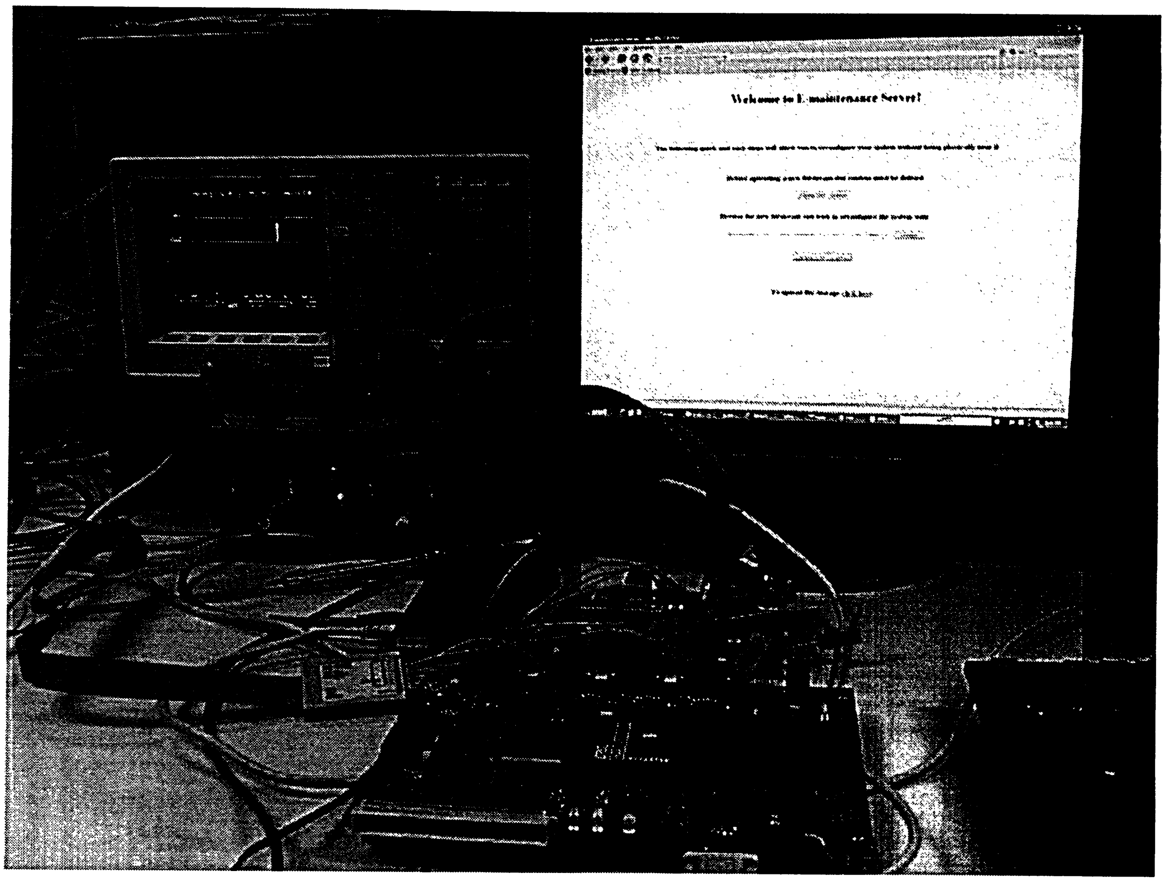
Appendix B1 – Photographic Image of e-Maintenance System Prototype connected to FPGA-based Platform



Appendix B2 – Photographic Image of System Setup for Measurements



Appendix B3 – Photographic Image of the Whole System Setup



Appendix C – Source Code for e-Maintenance System

```
/**********************************************************************
Author: Dina Goldenberg

Ryerson University, MEng. Project

July, 2007
**********************************************************************/

#class auto

#define TCPCONFIG 1

#define TCP_BUF_SIZE 2048

#define HTTP_MAXSERVERS 1
#define MAX_TCP_SOCKET_BUFFERS 1
#define HTTP_TIMEOUT          16

// serial buffer size
#define EINBUFSIZE  15
#define EOUTBUFSIZE 15
// serial baud rate
#define BAUD232 115200

// Iclude upload code
#define USE_HTTP_UPLOAD

// No name lookups required
#define DISABLE_DNS

#memmap xmem

// Sample library for PowerCoreFLEX series core modules
#use "PowerCoreFLEX.lib"

// The following MACRO's are preset for PowerCoreFLEX series core
modules
#define SF_SPI_CSPORT PDDR
#define SF_SPI_CSSHADOW PDDRShadow
#define SF_SPI_CSDD PDDDR
#define SF_SPI_CSDDSHADOW PDDDRShadow
#define SF_SPI_CSPIN 6

#use "sflash.lib"
#use "dcrtcp.lib"
#use "http.lib"

#ximport "Project/HTML/emaintenance_server.html"    index_html


char bufData[2048];
char buffer[256];
int pageNum;
int lastPage;
int bufLen;
int counter;


// This table maps file extensions to the appropriate "MIME" type.
This is
// needed for the HTTP server.
SSPEC_MIMETABLE_START
      SSPEC_MIME(".htm", "text/html"),
      SSPEC_MIME(".html", "text/html"),
      SSPEC_MIME(".gif", "image/gif"),
      SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END


int write_flash(int pageNum, char *buf)
{
   char buf2[256];

      printf("\nWriting to page %d ", pageNum);
    printf("%s", buf);
    printf("%s\n", buf + 128);
      sf_writeRAM(buf, 0, 256);
      sf_RAMToPage(pageNum);

   return 0;
}

int read_flash_page(int iNum, char buf)
{
      memset(buf, 0x00, sizeof(buf));
      printf("\nReading from flash page number %d\n", iNum);
      sf_pageToRAM(iNum);
      sf_readRAM(buf, 0, 256);

      return 0;
}


void delay(unsigned long wDelay)
{
      for (;wDelay>0;--wDelay);
}


int delete_slot_cgi(HttpState * s)
{
   int getC, delay100;

   getC = -1;
   delay100 = 0;

      // Open serial port E
    serEopen(BAUD232);
    serMode(0);

    // Clear serial buffers
```

```
     serEwrFlush();
     serErdFlush();

   printf("Delete Slot #1 - waiting 40s ...\n");
   serEputc(0x41);

   do
   {
      getC = serEgetc();
      delay(10);
      delay100++;
   } while (delay100 < 1000);

   if (getC == -1)
   {
      printf("No responce from the platform\n");
   }

   serEputc(0x65);
   delay(10000);
   serEputc(0x72);
   delay(10000);
   serEputc(0x73);
   delay(10000);
   serEputc(0x6C);
   delay(10000);
   serEputc(0x01);
   delay(10000);

   printf("Delay 40s...\n");
   delay(1000000);

      return 1;
}


int upload_cgi(HttpState * s)
{
   int sent, remain, getC, delay100;
   char serialBuf[256];
   int i;
   int temp;
   temp = 0;

   if (!http_getState(s))
   {
         http_setState(s, 1);
         printf("First entry:\n");
         printf("  HTTP version=%s\n",
            http_getHTTPVersion(s) == HTTP_VER_09 ? "0.9" :
            http_getHTTPVersion(s) == HTTP_VER_10 ? "1.0" :
            http_getHTTPVersion(s) == HTTP_VER_11 ? "1.1" : "unknown");
         printf("  HTTP method=%s\n",
            http_getHTTPMethod(s) == HTTP_METHOD_GET ? "GET" :
            http_getHTTPMethod(s) == HTTP_METHOD_POST ? "POST" :
            http_getHTTPMethod(s) == HTTP_METHOD_HEAD ? "HEAD" :
                                                       "unknown");
```

```
        printf("  Userid=%d\n", http_getContext(s)->userid);
        printf("  URL=%s\n", http_getURL(s));
  }

  switch (http_getAction(s)) {
  case CGI_START:
       bufLen = 0;
       pageNum = 0;
       lastPage = 0;
       getC = -1;
       delay100 = 0;

       memset(bufData, 0x00, sizeof(bufData));

       serEwrFlush();
       serErdFlush();

       printf("Start transmission to platform ...\n");
       delay100 = 0;
       serEputc(0x41);
       do
       {
            getC = serEgetc();
            delay(10);
            delay100++;
       } while (delay100 < 1000);

       if (getC == -1)
       {
           printf("No responce from the platform\n");
       }

       serEputc(0x6C);
       delay(10000);
       serEputc(0x64);
       delay(10000);
       serEputc(0x63);
       delay(10000);
       serEputc(0x72);
       delay(10000);
       serEputc(0x01);
       delay(10000);
       serEputc(0x00);
       delay(10000);
       serEputc(0x00);
       delay(10000);
       serEputc(0x00);
       delay(10000);

       printf("START content_length=%ld\n", http_getContentLength(s));
       // ContentLength - length of the whole uploaded file
       break;
     case CGI_DATA:
        printf("\nReceiving "); printf("%s\n", http_getData(s));
        if ((bufLen + http_getDataLength(s)) <= 256)
        {
           if (bufLen == 0)
        {
              memset(bufData, 0x00, sizeof(bufData));
              memcpy(bufData, http_getData(s),
                                       http_getDataLength(s));
        }
        else
              memcpy(bufData + bufLen, http_getData(s),
                                       http_getDataLength(s));

           bufLen = bufLen + http_getDataLength(s);
           if (bufLen == 256)
           {
                 // Send serial data
                 memset(serialBuf, 0x00, sizeof(serialBuf));
                 memcpy(serialBuf, bufData, 256);
                 sent = 0;
                 remain = 256;
                 while (sent!=256)
                 {
                     remain = 256 - sent;
                     sent += serEwrite(serialBuf, remain);
                 }
                 write_flash(pageNum, bufData);
                 pageNum++;
                 bufLen = 0;
           }
     }
     else // Too much for one flash page
     {
        // Fill up the buffer
        temp = 256 - bufLen;
        memcpy(bufData + bufLen, http_getData(s), temp);

        // Send serial data
        memset(serialBuf, 0x00, sizeof(serialBuf));
        memcpy(serialBuf, bufData, 256);
        sent = 0;
        remain = 256;
        while (sent!=256)
        {
           remain = 256 - sent;
           sent += serEwrite(serialBuf+sent, remain);
        }
        write_flash(pageNum, bufData);
        pageNum++;
        // Write the rest of the data to the buffer
        bufLen = http_getDataLength(s) - temp;
        memset(bufData, 0x00, sizeof(bufData));
        memcpy(bufData, http_getData(s) + temp, bufLen);
     }
     break;
  case CGI_END:
        printf("END ----------- actual received length=%ld\n",
                                 http_getContentLength(s));
        if (bufLen > 0)
        {
              memset(serialBuf, 0x00, sizeof(serialBuf));
```

```
                    memcpy(serialBuf, bufData, bufLen);
                    sent = 0;
                    remain = bufLen;
                    while (sent!=bufLen)
                    {
                       printf("Sending Serial data\n");
                       printf("%s", serialBuf+sent);
                       printf("%s\n", serialBuf+128);
                       remain = bufLen - sent;
                       sent += serEwrite(serialBuf+sent, remain);
                    }
                    lastPage = bufLen;
                    memset(bufData, 0x00, sizeof(bufData));
                    bufLen = 0;
                }
                for (i=0; i<16; i++)
                {
                    serEputc(0xFF);
                }
               printf("Final serial sent\n\n");
               // Close serial port E
               serEclose();
              break;
        case CGI_EOF:
              printf("EOF (unused content=%ld) \"%s\"\n",
                                 s->content_length, http_getData(s));
              http_switchCGI(s, "index.html");
              break;
        case CGI_ABORT:
              printf("ABORT!\n");
              break;
        default:
              printf("CGI: unknown action code %d\n", http_getAction(s));
              break;
    }

    return 0;
}


int download_cgi(HttpState* state)
{
        auto int i;

        printf(" substate=%d\n",state->substate);
        if(state->length)
        {
              // buffer to write out
              if(state->offset < state->length)
              {
                    state->offset += sock_write(&state->s,
                                state->buffer + (int)state->offset,
                                (int)state->length - (int)state->offset);
              }
              else
              {
                    state->offset = 0;
                    state->length = 0;
              }
      }
      else
      {
         switch(state->substate)
         {
         case 0:
              strcpy(state->buffer, "HTTP/1.0 200 OK\r\nContent-type:
 *//**\r\n\r\n");
              state->length = strlen(state->buffer);
              state->offset = 0;
              state->substate++;
              counter = 0;
           break;

         case 1:
               if (counter<=pageNum)
               {
                    read_flash_page(counter, &buffer);
                    memset(buffer, 0x00, sizeof(buffer));
                    sf_pageToRAM(counter);
                    if (counter == pageNum)
                    {
                       sf_readRAM(buffer, 0, lastPage);
                       memcpy(state->buffer, buffer, lastPage);
                       state->length = lastPage;
                    }
                    else
                    {
                       sf_readRAM(buffer, 0, 256);
                          memcpy(state->buffer, buffer, sizeof(buffer));
                          state->length = 256;
                    }
                    counter++;
                    state->main_timeout=set_timeout(HTTP_TIMEOUT);
               }
               else
                     state->substate++;

             break;

             default:
                   state->substate = 0;
                   printf(" Terminating !!!!\n");
                   return 1;
             }
        }
        printf("\nreturning zero %d\n",state->substate);
        return 0;
}

// The flash resource table is now initialized with these macros...
SSPEC_RESOURCETABLE_START
      SSPEC_RESOURCE_XMEMFILE("/index.html", index_html),
      SSPEC_RESOURCE_CGI("upload.cgi", upload_cgi),
      SSPEC_RESOURCE_FUNCTION("/file.bin", download_cgi),
      SSPEC_RESOURCE_FUNCTION("/DeleteSlot.cgi", delete_slot_cgi)
SSPEC_RESOURCETABLE_END


void main()
{
   char buf[20];
   unsigned long i;
   int n;

   // Initialize I/O to use PowerCoreFLEX prototyping board
   brdInit();

   //Initialize SPI driver for use with serial flash.
   sfspi_init();
   if(sf_init())  //Initializes serial flash chip.
   {
      printf("Serial flash initialize error!");
      exit(1);
   }
   else
   {
        printf("Flash init OK\n");
   }

   memset(bufData, 0x00, sizeof(bufData));
   sock_init();
   http_init();
   tcp_reserveport(80);

   pageNum = 0;
   lastPage = 0;
   bufLen = 0;

   i=0;
   n=0;

   while (1)
   {
      http_handler();
   }
}
```

Appendix D – HTML Source

```
<html>
<head><title>E-Maintenance Server</title></head>
<body>

<P>
<Br>
<H1 align=center>Welcome to e-Maintenance Server!</H1>
<Br>
<P>

<Br>
<H3 align=center>The following quick and easy steps will allow you to
 reconfigure your system without being physically near it</H3>
<Br>

<H3 align=center>Before uploading a new bitstream slot content must be
 deleted </H3>

<H1 align=center><FORM ACTION="DeleteSlot.cgi" METHOD="POST">
<INPUT TYPE="SUBMIT" VALUE="Delete Slot Contents"><Br>
</FORM></H1>
<P>

<H3 align=center>Browse for new bitstream you wish to reconfigure the
 system with</H3>
<H1 align=center><FORM ACTION="upload.cgi" METHOD="POST"
 enctype="multipart/form-data">
<TABLE BORDER=0 CELLSPACING=2 CELLPADDING=1>

<TR>
        <TD><INPUT TYPE="FILE" NAME="/A/new.htm" SIZE=70></TD>
<TR>
</TABLE>

<P>
<P>
<INPUT TYPE="SUBMIT" VALUE="Upload New Firmware"><Br>
</FORM></H1>

<Br>
<H3 align=center>To download file storage <a href="file.bin">click here</a></H3>

</body>
</html>
```