2011

Medium Access Control Layer Implementation on Field Programmable Gate Array Board for Wireless Networks

Ayman Ramzi Alghamdi

Medium Access Control Layer Implementation on Field Programmable Gate Array Board for Wireless Networks

(Spine title: MAC Layer Implementation on FPGA for WN)

(Thesis format: Monograph)

by

Ayman Ramzi Alghamdi

Graduate Program
in
Engineering Science
Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master in Engineering Science

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Certificate of Examination

THE UNIVERSITY OF WESTERN ONTARIO
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES
CERTIFICATE OF EXAMINATION

Chief Advisor:

Dr. Abdallah Shami

Advisory Committee:

Examining Board:

Dr. Roger Khayat

Dr. Abdelkader Ouda

Dr. Raveendra Rao

The thesis by
Ayman Ramzi Alghamdi
entitled:
Medium Access Control Layer Implementation on Field Programmable Gate Array Board for Wireless Networks
is accepted in partial fulfillment of the
requirements for the degree of
Master in Engineering Science

Date: _____

Chair of Examining Board
Dr. K. Adamiak

Abstract

Triple play services are playing an important role in modern telecommunications systems. Nowadays, more researchers are engaged in investigating the most efficient approaches to integrate these services at a reduced level of operation costs. Field Programmable Gate Array (FPGA) boards have been found as the most suitable platform to test new protocols as they offer high levels of flexibility and customization. This thesis focuses on implementing a framework for the Triple Play Time Division Multiple Access (TP-TDMA) protocol using the Xilinx FPGA Virtex-5 board. This flexible framework design offers network systems engineers a reconfigurable platform for triple-play systems development.

In this work, MicorBlaze is used to perform memory and connectivity tests aiming to ensure the establishment of the connectivity as well as board's processor stability. Two different approaches are followed to achieve TP-TDMA implementation: systematic and conceptual. In the systematic approach, a bottom-to-top design is chosen where four subsystems are built with various components. Each component is then tested individually to investigate its response. On the other hand, the conceptual approach is designed with only two components, in which one of them is created with the help of Xilinx Integrated Software Environment (ISE) Core Generator. The system is integrated and then tested to check its overall response.

In summary, the work of this thesis is divided into three sections. The first section presents a testing method for Virtex-5 board using MicroBlaze soft processor. The following two sections concentrate on implementing the TP-TDMA protocol on the board by using two design approaches: one based on designing each component from scratch, while the other one focuses more on the system's broader picture.

Acknowledgements

Dr. Abdallah Shami - For everything you have offered and your continuous support.

Dr. Roger Khayat,
Dr. Abdelkader Ouda,
Dr. Raveendra Rao,
Dr. Kazimierz Adamiak - For being in the Examination Board.

Dan Dechene,
Christopher Kennedy,
Tomasz Rybak,
Mahadevan Balakrishnan - For your feedback.

Thomas Daniel Wallace,
Abdelkader Abdessameud,
Abdou Ramadan Ali Ahmed,
Maysam Mirahmadi,
Andrew Roberts - For being an amazing labmates.

Ramzi and Azzah Alghamdi - This wouldn't be achieved without you, thanks for being the best parents ever.

Table of Contents

List of Tables

List of Tables

List of Figures

Acronyms

AC	*Access Categories*
ACE	*Advanced Configuration Environment*
ADC	*Analog to Digital*
AGC	*Apollo Guidance Computer*
AHB	*Advanced High performance Bus*
AP	*Access Point*
APB	*Advanced Peripheral Bus*
ARQ	*Automatic Repeat Request*
ASIC	*Application-specific Integrated Circuit*
BPI	*Byte-wide Peripheral Interface*
BS	*Base Station*
BSP	*Board Support Package*
BSB	*Base System Builder*
C/A	*Coarse/Acquisition*
CDMA	*Code Division Multiple Access*
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
CSMA	*Carrier Sense Multiple Access*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
CSMA/ECA	*Carrier Sense Multiple Access with Enhanced Collision Avoidance*
CT	*Computed Tomography*
CTS	*Clear to Send*
CW	*Contention Windows*
DAC	*Digital to Analog*
DCF	*Distributed Coordination Function*
DCO	*Digitally Controlled Oscillator*
DDR	*Double Data Rate*

DIFS	*Distrusted Inter-frame Space*
DIUC	*Downlink Interval Usage Code*
DMA	*Direct Memory Access*
DRM	*Downlink Request Management*
DRMP	*Dynamically Reconfigurable MAC Processor*
DSP	*Digital Signal Processing*
DSSS	*Direct-sequence Spread Spectrum*
DVI	*Digital Visual Interface*
EDCA	*Enhanced Distributed Channel Access*
EDCF	*Enhanced Distributed Coordination Function*
EDK	*Embedded Development Kit*
ELF	*Executable Linking Format*
FDM	*Frequency-Division Multiplexing*
FPGA	*Field Programmable Gate Array*
FSL	*Fast Simplex Link*
FSM	*Finite State Machine*
FSU	*Frame Scheduling Unit*
GMII	*Gigabit Media Independent Interface*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
I/O	*Input/Output*
ISE	*Integrated Software Environment*
JTAG	*Joint Test Action Group*
LMB	*Local Memory Bus*
MAC	*Medium Access Control*
MPLB	*Master Processor Local Bus*
MRI	*Magnetic Resonance Imaging*
MDM	*MicroBlaze Debug Mode*
MRX	*MAC Receiver module*
MSC	*Message Sequence Chart*
NSS	*Network Switching Subsystem*
OFDM	*Orthogonal Frequency Division Multiplexing*

OPB	*On-chip Peripheral Bus*
PLB	*Processor Local Bus*
PLCP	*Physical Layer Convergence Procedure*
PMD	*Physical Medium Dependent*
PROM	*Programmable Read Only Memory*
QoS	*Quality of Service*
RFU	*Reconfigurable Function Unit*
RGMII	*Reduced Gigabit Media Independent Interface*
RISC	*Reduce Instruction Set Computer*
RTOS	*Real Time Operating Systems*
RTS	*Request to Send*
RX	*Receiver*
SANE	*Self Adaptive Networked computing Element*
SCSA	*Single-Carrier Scheduling Algorithm*
SDK	*Software Development Kit*
SDL	*Specification and Description Language*
SDM	*Spatial Division Multiplexing*
SDR	*Software Defined Radio*
SG	*Scheduling Group*
SGMII	*Serial-GMII*
SIFS	*Short Inter-frame Space*
SNR	*Signal to Noise Ratio*
SoC	*System-on-chip*
SPI	*Serial Peripheral Interface*
SPLB	*Slave Processor Local Bus*
SS	*Subscriber Station*
SSRAM	*Synchronous Static Random-access Memory*
TDM	*Time-Division Multiplexing*
TEMAC	*Tri-mode Ethernet MAC*
TP-TDMA	*Triple-Play Time Division Multiple Access*
TX	*Transmitter*
TXOP	*Transmission Opportunity*

UART	*Universal Asynchronous Receiver-Transmitter*
UGS	*Unsolicited Grant Service*
URMA	*Uplink Request Management Agent*
UWB	*Ultra-Wideband*
VOIP	*Voice over Internet Protocol*
WEP	*Wired Equivalent Protection*
WLAN	*Wireless Local Area Network*
WRR	*Weighted Round Robin*
XML	*Extensible Markup Language*
XPS	*Xilinx Platform Studio*

Chapter 1
Introduction

In the introduction chapter, the reader is exposed to the motivations behind the work implemented in this thesis, the contributions, and the thesis organization. This chapter is divided into three sections; the first section discusses the importance of triple-play services from a business point of view in terms of costs and effectiveness as a main research motivator. The second section presents the methods followed and the contributions achieved in this work. While the last section outlines the thesis organization in further detail.

1.1 Research Motivation

The main drive for the work done in this thesis is simply summarized in two aspects. The first aspect can be seen through the recent increasing demand for triple play services being a reason for researchers to address this important field. Second, microcontrollers-based embedded systems allow easy modifications to the system's configurations.

Recent advances in new telecommunication systems has led traditional markets to change the way they provide their services; the isolated standalone services approach is no longer applicable; services integration is now becoming the keyword for network vendors [1]. However, technological limitations can be a barrier that slow down the integration of these services implementation; limitations include bandwidth coverage and bit-rate [2].

Looking at the Internet Protocol (IP) networking world, vendors are competing to fulfill the market's demand for triple-play products and platforms, while service providers are being aggressive in marketing their new multimedia services. Video over IP is attracting much attention recently in two different forms: television (TV) channels being transmitted to customers over the IP network (IPTV) and Video on demand (VoD) in which users can request their favorite TV shows to be streamed over the IP network [3].

With more focus on video communications, we can see that the video conferencing market is facing a Perfect Storm [4]. This huge demand on video conferencing is driven by three factors: the advances in endpoint technologies, high expectations of users, and, finally, the improvements achieved in terms of speed, cost and network

availability. All these factors are integrated together to speed up the adoption of video conferencing in business environments. This huge demand could be seen in the rapid growth of leading vendors in terms of revenues, shipments, and profits [5].

Telecommunication companies have never been more interested in providing TV entertainment services as they are now. In fact, they are not only interested in this market because of its profit, they actually have no choice but to get themselves involved in the competition since cable companies are starting to dominate what telecommunication companies used to run, i.e., voice and data services. Therefore, offering the three services combined is the only way to keep customers loyal to their telecommunication providers [6]. Both cable and telecommunication companies provide triple-play services; however, they don't use the same last mile approach infrastructure. Telecommunication providers use mainly Digital Subscribe Lines (DSL), while cable carriers use Hybrid Fiber Coaxial (HFC) to reach a customer's home.

Major operators are interested in next generation IP-based networks as they offer the most cost-effective and future-proof platforms to deliver triple-play services to their clients. That approach also will provide significant advantages in terms of building and maintaining costs when compared to the parallel networks approach (i.e., voice network, video network, and data network). Moreover, a study by Royal Bank of Canada (RBC) Capital Markets suggests that 48% of Americans who own flat screens are willing to buy cable TV from their telecom company rather than getting cable TV from another company [7].

Configurable embedded systems are often referred to as FPGAs. They are considered future-oriented building bricks which allow a high level of customization of the hardware at reasonable costs. This makes them an effective factor for time-to-market by avoiding expensive redesigning of the board. Software modifications on the design will make the board ready to operate with changes [8].

1.2 Methods and Contributions

This section focuses on the methods followed and contributions achieved in this thesis. Mainly, two methods were followed to investigate the proposed ideas: the first method was to use a soft processor called MicroBlaze; while the second method was to write Very-high-speed integrated circuit Hardware Description Language (VHDL) codes in the ISE programming environment. Obviously, both products, MicroBlaze and ISE, are Xilinx property since the targeted board is also Xilinx's. Through out this work, three contributions could be stated. The first contribution was achieved using MicroBlaze soft processor to test the board; while the other two contributions were more focused on designing schedulers for TP-TDMA.

The decision to use MicroBlaze to create a soft processor was not only made to test the board; in fact, it was also a try to investigate the possibility of using such a processor in designing a soft scheduler, which could be a good start for future work.

MicroBlaze was the hardware platform used to establish communication between the computer and the FPGA. By establishing this communication, the board was actually made ready to receive any set of instructions to deploy certain algorithms. Using MicroBlaze to test the board was only a starting point for the more expanded ambition of deploying scheduling algorithms using this soft processor. In addition to that, MicroBlaze is attractive to use since it is programmed using C language; which in turn brings a wider range of programmers in.

To implement a TP-TDMA scheduler, the two approaches were investigated using VHDL on ISE. The first approach was implemented through creating a complete bottom-to-top system design, then testing each component in the system individually. In this design, the system was constructed with four subsystems, each with a different number of components. The components were designed using VHDL, and tested by creating a test bench for each; inputs were fed individually to each component to observe the component's response. The second approach could be thought of as being more of an integrated design. Two subsystems only were designed: the Ethernet Controller and the Scheduler. The Ethernet Controller used the advantage of the easy implementation of ISE Core Generator components; on the other hand, the Scheduler was designed using a Finite State Machine (FSM).

1.3 Thesis Organization

This thesis is composed of seven chapters. While the first chapter introduces the thesis, the remainder of this thesis is organized as follows:

- Chapter II provides a comprehensive background on the two main topics that this thesis covers. In the first section, Medium Access Control (MAC) layer protocols are investigated thoroughly by looking into the carrier sense multiple access as well as the collision-free protocols. The second section reviews embedded systems, with further details on the history of embedded systems and characteristics. After that, Xilinx hardware and software embedded systems are investigated.

- Chapter III aims to take the reader through a literature review covering two aspects; the Virtex 5 board and MAC layer implementation on boards. The first section provides a detailed description of the board's FPGA as well as its peripherals. In the second section, a literature review is presented for MAC layer implementation on FPGA boards and other boards, as well as some highlights on the MicroBlaze processor.

- Chapter IV discusses the proposed system design in three sections. In the first section, the board is tested using MicroBlaze, while in the next two sections a scheduler design is proposed using two different approaches: Systematic and Conceptual.

- Chapter V presents the results obtained from the work done in chapter four. Similarly, it's divided into three sections; each section presents the results achieved followed by further discussions.
- Chapter VI mainly provides a foundation for future work. The purpose of this chapter is to provide the starting step for designers who seek to implement scheduling tasks in general.
- Chapter VII concludes the thesis work by presenting the main observations made throughout this work.

Chapter 2
Background

Implementing Medium Access Control (MAC) layer protocols on Field Programmable Gate Array (FPGA) boards requires a comprehensive understanding of the MAC layer's functionality and protocols. The MAC layer's functionality simply includes accomplishing two main tasks: packet addressing and channel access control. The MAC layer is often referred to as a sub-layer since it is considered the interface between the Logical Link Control sub-layer and the network's physical layer. MAC protocols are implemented on hardware that is usually called the Medium Access Controller. Manufacturers of MAC controllers have benefited from recent advances in embedded systems technology allowing them to have a wide range of controllers on both single programmed boards as well as re-programmable boards.

This chapter will review the Medium Access Control (MAC) layer and embedded systems technologies. In the MAC layer section, carrier sense protocols are investigated including collision-detection and avoidance protocols. Also, collision-free protocols are studied through different protocols designed for IEEE 802.16 and IEEE 802.11 technologies. In the embedded systems section, the history of these systems and their characteristics are briefly studied. After that, a detailed review of Xilinx hardware and software is presented, followed by a quick review of the MicroBlaze processor.

2.1 MAC Layer

Network links are divided into two types: point-to-point link and broadcast link. A broadcast link can have multiple connected nodes sending and receiving using the same shared media. In such environments, the problem of choosing which node has the right to send/receive arises.

Multiple Access protocols are a set of protocols used to regulate the use of the shared media between nodes in the same network. Since all nodes in the network can transmit their frames simultaneously, some nodes may transmit at the same moment. As a result, the transmitted frames will collide. The collision problem leads to losing the frames involved in the collision, and to wasting the media's bandwidth since the frames were not transmitted successfully. For that, it is of high importance to arrange transmission between active nodes in the network in order to utilize the bandwidth efficiently. Many multiple access protocols have been suggested; however, they all

can be classified into one of three categories: channel partitioning protocols, random access protocols, and taking-turns protocols.

Channel partitioning protocols are the protocols in which each node is assigned a dedicated transmission slot; Time-Division Multiplexing (TDM), Frequency-Division Multiplexing (FDM), and Code Division Multiple Access (CDMA) are examples of this class. In TDM, nodes are assigned a dedicated time slot, while in FDM nodes are assigned frequency slots. TDM and FDM share the same advantages and drawbacks. Both manage to avoid collisions by dividing the bandwidth fairly between the nodes, yet both limit the nodes in the network to a fixed bandwidth. This drawback is clearly seen as a principle disadvantage when only one node is active, i.e, bandwidth misusage. CDMA, on the other hand, avoids that problem by assigning different codes to each transmitting node. Having different codes, the network can transmit frames simultaneously between its nodes. Obviously, the receiver node should recognize the transmitter's code.

Compared to channel partitioning protocols, random access protocols give the transmitting node the channels' full bandwidth, and in case that collisions happen, the nodes involved will retransmit their frames until all frames have been successfully transmitted. Slotted ALOHA is a simple example of random access protocols. In slotted ALOHA, all frames are of the same size, time is divided into fixed slots, nodes can send their frames only at the beginning of a slot, and nodes are synchronized to recognize the beginning of a slot. When a node transmits its frame and detects a collision, the node retransmits its frame with a probability p in each slot after collision occurred. Although slotted ALOHA has the advantage of being a decentralized system, it has a weak efficiency that can't exceed 37% of the channel bandwidth when there is a large number of nodes connected.

Another example of the random access protocols is the Carrier Sense Multiple Access (CSMA) protocol. As it can be understood from its name, this protocol has a carrier sense feature in which the node listens to the channel before sending a message. If the channel is busy, the node waits for a random time, and then checks the channel's availability again. The node will send only when the channel is idle. Two CSMA protocols are well investigated in the literature: CSMA with Collision Detection (CSMA/CD) and CSMA with Collision Avoidance (CSMA/CA). CSMA/CD is the media access protocol used in Ethernet, while CSMA/CA is the protocol used in wireless networks such as IEEE 802.11 [9].

2.1.1 Carrier Sense Multiple Access Protocols

As mentioned earlier, CSMA protocols are widely used in wired and wireless networks. In fact, they use similar approaches to reduce the possibility of having a collision. Now, we will explore the CSMA/CD protocol that is widely deployed in Ethernet networks. Then, we will focus on the CSMA/CA protocol, which will be followed by an enhanced version of the collision avoidance protocol.

2.1.1.1 Carrier Sense Multiple Access with Collision Detection

In CSMA/CD, a node will first check the channel availability; once the node verifies that the channel is idle, it'll send its frame. While transmitting its frame, the node will keep listening to the channel. If a collision is detected, it will stop its transmission; then it will re-transmit using algorithms to calculate the waiting time (back off). The need for collision detection arises because of the channel propagation delay. When a node verifies that the channel is idle, it doesn't guarantee that a collision will not occur. For example, while node A is transmitting a frame to node B, it is possible that node C has not yet received this transmission as a result of the propagation delay. Consequently, node C will assume that the channel is idle, and might start sending its frame leading to a collision. Therefore, CSMA/CD ceases transmission once a collision is detected to improve the network's transmission performance.

CSMA/CD uses an algorithm called exponential back off to calculate the waiting time for re-transmitting in Ethernet. After the *nth* collision, the node chooses a random value K such that:

$$\{0, 1, 2, 2^m - 1\}, where$$

$$m = min(n, 10).$$

The waiting time is *K*512*. When collisions are detected, the node that detected it transmits a 48 bit jamming signal. The purpose of the jam signal is to inform all nodes sharing the media that a collision has occurred.

MAC protocol in 802.11 differs from the one in Ethernet in two main aspects. First, 802.11 networks use CSMA/CA. Second, 802.11 networks use link layer acknowledgment because of the frequent bit errors of wireless physical channels. In fact, there are two important reasons why CSMA/CD is not suitable for wireless networks. The main reason is that CSMA/CD requires a dual detection ability, i.e., the ability to send and receive at the same time, which is not possible in wireless networks because of the huge difference in the signal strength of received and sent signals. The other reason is the hidden terminals problem in wireless networks, which makes the node unable to detect all collisions and transmissions. Because the 802.11 does not use CSMA/CD, 802.11 networks transmit the entire frame at once.

2.1.1.2 Carrier Sense Multiple Access with Collision Avoidance

Suppose a node in 802.11 has a frame to send, using CSMA/CA; the process will be as follows. The node will sense the media to check its availability; if the media is idle, the node transmits its frame after a period known as the Distrusted Inter-frame Space (DIFS). If the media is busy, the node chooses a random back off value and starts counting down to detect when the media is idle. The node remains idle until the counter reaches zero, and then starts transmitting its frame. After that, the sending node will wait for an acknowledgment from the receiving node. The receiving node

waits for a period known as the Short Inter-frame Space (SIFS) then sends back an acknowledgment. If the acknowledgment is received, the transmission is considered complete; otherwise, the sending node returns to a back off phase with a larger value.

In CSMA/CA, it is to be noted that even if the channel is idle, the node will have to wait for the counter to reach zero in order to start transmitting. The reason for that is to reduce the possibility of having a collision by letting each node to wait for a different amount of time. However, a collision might still occur in a wireless network as a result of the hidden terminals or a similar chosen back off time.

In order to avoid the hidden terminal problem, 802.11 MAC protocol suggests using Request to Send (RTS) and Clear to Send (CTS) control frames. When a node is about to transmit a frame, it first sends an RTS frame to the Access Point (AP) mentioning the required time for both of its frame of data and frame of acknowledgment. Then the AP sends a CTS frame to all the nodes connected to it in order to make all nodes aware of the transmission and give permission to the sending node to start transmitting. RTS and CTS frames not only help to resolve the hidden terminal problem, but also they ensure a clear transmission of the data and acknowledgment frames.

2.1.1.3 Carrier Sense Multiple Access with Enhanced Collision Avoidance

Since it's a lightweight and decentralized protocol, CSMA/CA is suitable for IEEE 802.11 networks. However, CSMA/CA does not utilize the transmission history in stations that have many queue of frames to send. In other words, the protocol can employ previous transmission attempts to effectively reduce the number of collisions in subsequent transmissions. This addition to the CSMA/CA is referred to as CSMA with Enhanced Collision Avoidance (CSMA/ECA). After a transition phase, CSMA/ECA is able to offer a collision-free access protocol. It also works fairly with the existing CSMA/CA and is easy to implement without further computational modifications.

The channel time in CSMA/CA is divided into three slot types: empty, successful, and collision. A slot belongs to the empty category if there is no frame to transmit; it belongs to the successful category if there is only one transmitted frame; and it belongs to the collision category if there is more than one transmitted frame. In the empty and collision slots, the channel time is considered wasted. After a collision, the station has to wait (back off) in order to randomly choose another back off time. CSMA/ECA suggests choosing a deterministic back off time after a successful transmission in which the number of active stations is less than the value of the back off time. Conversely, the legacy CSMA/CA chooses a random back off time even after a successful transmission.

This slight modification in the CSMA/CA protocol led to a remarkable improvement in channel utilization. To verify that, [10] examined two simulation scenarios;

one in which half of the stations used CSMA/ECA while the other half used the legacy CSMA/CA; another simulation was performed with stations using CSMA/ECA protocol only. In both scenarios, the channel utilization improved; however, the channel utilization's improvement using pure CSMA/ECA was 0.8 to 1 compared to an average of 0.8 in the first scenario. The reduction in the number of possible collisions was the reason for this improvement. Even though CSMA/ECA is likely to work as a collision-free system, it could still result in a collision in one of two situations: the entrance of a new station, or a channel error.

When a new station tries to start transmitting, i.e., become an active node, it may push the system back to the transition phase if its first transmission resulted in a collision. To avoid this, a smart entry approach can be deployed. A station should keep track of the empty slots in order to schedule its first transmission. In case there are no empty slots, the station should postpone its transmission until there is slot availability.

In a case where the channel is affecting the transmission reliability, CSMA/ECA performance can also be affected. CSMA/ECA deals with channel error in a similar manner as if it were a collision. It is to be noted that the affect of the channel error is greater on CSMA/ECA than it is on the legacy CSMA/CA. However, in comparing CSMA/CA performance under channel error to CSMA/ECA, the enhanced version is still better than the legacy version.

Other studies have used a similar approach to CSMA/ECA. In [11], an enhancement was introduced to the IEEE 802.11 protocol called EBA. In this protocol, two fields are added to the MAC headers to inform the other stations about the back off value. Even though this approach reduces the rate of collisions, it requires modifications of the MAC headers leading to further complications.

2.1.2 Collision-free Protocols

After discussing CSMA protocols, we will now focus on collision-free protocols. In this section, the protocols for IEEE 802.16 and IEEE 802.11 will be investigated: the Single Carrier Scheduling Algorithm and Triple Play Time Division Multiple Access.

2.1.2.1 Collision-free Protocols for IEEE 802.16

The authors of [12] propose an uplink bandwidth allocation scheme for polling services where the Markov modulated Poisson process is applied. The limitation of this scheme is that it assumes a fixed bandwidth allocation at one Subscriber Station (SS) and considers only the outbound transmission scheduling. The drawback of this scheme is that it eliminates the consideration of a bandwidth request for each connection. In [13], a packet scheduling algorithm is introduced where fixed allocation, earliest deadline first, weighted fair queuing, and equal sharing schemes are applied. To achieve ultimate Quality of Service (QoS), this algorithm requires intelligent outbound

transmission scheduling at each SS since the bandwidth allocated to an SS is an aggregated grant for all the connections at the SS. Another QoS scheme is proposed by [14]; in this scheme, the BS bandwidth allocation and the outbound transmission scheduling are addressed. To provide QoS at the connection level, many functions are introduced but the strict QoS parameters are not considered.

2.1.2.2 Single Carrier Scheduling Algorithm for IEEE 802.16

IEEE 802.16 standard supports QoS allowing the service categorization to be decided by the vendor. However, in the implementation of QoS specific issues must be addressed to achieve reliable and link-adaptive high rate transmission over the wireless channel. These issues include informing the BS about the connection level bandwidth for each SS; properly allocating the wireless resources among all SSs; and scheduling the transmissions over the shared channel for all SSs in such a way as to meet ter QoS requirements.

MAC scheduling services are differentiated into four types in order to accommodate different service applications. The first type is the Unsolicited Grant Service (UGS), which supports real-time applications that require fixed-size periodical data packets such as T1/E1. This type of MAC scheduling is given QoS with a dedicated traffic rate, maximum latency, and tolerated jitter. Real-time Polling Service (rtPS) is the second type of MAC scheduling in which periodical variable-sized data packets such as Moving Pictures Experts Group (MPEG) are supported. The QoS requirements for this type are: minimum traffic rate dedication, maximum sustained traffic rate, and maximum latency. The third type is the Non-real-time Polling Service (nrtPS), which supports non periodical variable-sized data that requires a minimum traffic rate such as File Transfer Protocol (FTP) applications. For this type, the QoS requirements are: minimum dedicated traffic rate, and maximum sustained traffic rate. Best Effort (BE) is the fourth type where data that doesn't require a minimum service level is supported. This type's QoS requires only minimum sustained traffic rate.

The UGS type is the only type that is delay sensitive among all the four types of MAC scheduling. As a result, the bandwidth dedicated to the BS for UGS is periodically fixed. Downlink traffic can be also categorized using the same four scheduling types; however, since the BS has all the downlink traffic information, the scheduling design of the uplink traffic is the only challenge to be faced.

In designing an efficient QoS control scheme, several aspects have to be considered. The BS should divide the downlink and the uplink frames properly to ensure that each SS uplink transmission window to meets its QoS requirements. Also, the BS should be informed periodically of how much bandwidth is required for each SS connection; this will help the BS manage its radio resources more efficiently. Moreover, the signaling overhead should be reduced for each connection bandwidth request. With these design aspects taken into consideration, the MAC protocol should be able

to provide a guaranteed QoS for each MAC scheduling service type, optimize a BS's awareness about each connection bandwidth requirement, and reduce the operational overhead for each bandwidth request.

A proposed QoS control scheme for IEEE 802.16 called Single-Carrier Scheduling Algorithm (SCSA) [15] consists of two blocks: the Uplink Request Management Agent (URMA) and the Frame Scheduling Unit (FSU). The URMA is located at each SS and serves to communicate to the BS its SS connections bandwidth requests with minimum signaling overhead, and to assist in scheduling uplink transmissions at the SS. On the other hand, the FSU is located at the BS where it's responsible for collecting information about each connection's bandwidth needs in the network; performing resource allocations for each SS; defining a new frame based on the resource allocation, and assisting to schedule downlink transmissions at the BS. It can be noticed that each SS URMA performs some functionalities locally at the SS instead of at the BS in order to reduce the overhead required for bandwidth requests.

The URMA consists of three modules: the service measurement module, the QoS enforcement module, and the SS request generation module. The service measurement module calculates the instant bandwidth request of each connection at the end of each uplink transmission window of the SS; that is done considering the connection's queue length and the MAC headers required to transmit the backlogged traffic. The QoS enforcement module maintains a QoS timer running at the SS for each rtPS and nrtPS connection. This timer is synchronized with the SS's system clock, where the measurement rate for a connection should be equal to the minimum reserved traffic rate.

The QoS enforcement module performs its operation in two steps. First, it divides the bandwidth request into two portions: bandwidth guaranteed and non-bandwidth guaranteed. This division is made for rtPS and nrtPS connections while BE connections are always given non-bandwidth guaranteed. Second, it further divides the bandwidth guaranteed portion into imminent and non-imminent parts depending on the maximum latency of the connection. The SS request generation module checks the service type of connections running at the SS and the output of the QoS enforcement module, then it generates three bandwidth requests for each SS.

At the BS end, FSU generates a new frame once it receives the prioritized bandwidth requests. To perform that step, three functional modules are involved: the Downlink Request Management (DRM) module, the resource allocation module, and the frame creation module. The DRM module acts in a similar manner to the URMA at the SS except that the downlink connections with the same Downlink Interval Usage Code (DIUC) are grouped together in the next frame. Each connection shares a common set of prioritized bandwidth requests and is referred to as a Scheduling Group (SG). For each SG, the resource allocation module allocates the transmission capacity according to its prioritized bandwidth request. After that, symbol assignments are made by converting the prioritized bandwidth requests of each SG into symbol needs. The last module in FSU, the frame creation module, is

translating the symbol assignments from the resource allocation module into timing information in terms of physical slots to create the new frame.

The SCSA scheme shifts some functionalities formerly performed by the BS to the SS in order to reduce signaling overhead. This helps to guarantee providing each connection with the QoS required. Also, with a cross-layer design for resource allocation, this scheme is capable of resisting wireless link degradation.

2.1.2.3 Triple-Play TDMA for IEEE 802.11

IEEE 802.11e networks use the Enhanced Distributed Coordination Function (EDCF) protocol to deliver triple-play services, i.e., voice, video, and data. However, this protocol is not capable of utilizing the channel resources efficiently [16]. When the network is at its full load, EDCF is able to achieve only 45% (as throughput) of the total dedicated bandwidth. The reason for such a waste in the bandwidth can be attributed to packets retransmission due to channel errors or collision, physical layer overhead, MAC layer overhead, and the contention mechanism. Therefore, EDCF protocol is not suitable for supporting triple-play services.

In order to meet triple-play services requirements by eliminating collisions and reducing MAC layer overhead, a Triple-Play Time Division Multiple Access (TP-TDMA) customized protocol is proposed to replace the 802.11e EDCF. This customized protocol is designed to meet specific system requirements; an Access Point (AP) connected to a cabled-hub with four subscriber stations as the network clients. TP-TDMA suggests a reduced header size to reduce the MAC layer header and an automatic repeat request (ARQ) to select the frame level. Packets are retransmitted in TP-TDMA based on the traffic priority, whereas voice is not because of time-sensitivity. The TP-TDMA downlink allows subscriber stations to downstream video, voice, and data while the uplink allows subscribers to upstream data and voice.

In the TP-TDMA protocol, the frames are structured as follows: the downlink frame consists of a beacon packet followed by four voice slots; then video and data traffic are divided dynamically in the remaining four slots. The uplink frame is also divided into four dynamically sized slots for the data traffic followed by four slots utilized for uplink voice traffic. Guard-intervals are used in order to avoid a collision between consecutive slots on both the downlink and uplink frames. These guard-intervals are also added to separate subscribers' uplink slots. It is important to mention that each station of the four subscriber stations is assigned two slots during both the uplink and downlink frames.

TP-TDMA suggests a modified MAC header keeping the first two fields (the first four bytes) as they are in the legacy 802.11e. After those two fields, a Frame Control field of two bytes is added to denote the type of packet and the other packet fields. Receiver and Transmitter Address fields of one byte for each are then followed. In addressing those fields, the first four bits are assigned to the network id while the remaining four bits are assigned to the host id. The AP is reserved a unique host

address of 0x00 while subscriber stations are identified using particular bits for each station. Addressing fields are followed by sequencing fields; two fields each of which have two bytes are Video Sequence and Best-Effort Sequence, respectively. Similarly, those sequencing fields are also followed by two ARQ bitmap fields each of which has four bytes. Both sequencing and ARQ bitmap fields are used to support the retransmission. A Feedback field following the ARQ bitmap fields is used to provide information about the size of subscribers' data buffer.

For efficient management of the wireless resources, TP-TDMA uses a dynamic bandwidth allocation scheme in which transmissions are scheduled based on traffic class. Slots are allocated with a fixed length in the downlink and uplink frames for the voice channel in order to provide a guaranteed delivery of voice traffic. Then, video and data packets, which are scheduled downstream for retransmission, are allocated the channel's maximum number of retransmissions. This number can be configured separately for each class of traffic in such a way that any packet that exceeds this number will be discarded. Video stream traffic is scheduled using an algorithm in which downstream traffic is allocated based on the number of buffered packets for a specific station.

Packets that require retransmission are handled using a Selective-ARQ mechanism. While video and data are controlled by ARQ bitmap fields in the MAC headers, voice traffic does not use ARQ mechanism. The AP maintains different ARQ bitmaps for each station and traffic class. The ARQ bitmap size should not exceed the maximum number of packets of a given class and destination transmitted during a frame transmission.

When comparing TP-TDMA to the legacy EDCF protocol, it can be found that the bound on frame delay of TP-TDMA is four times smaller than EDCF. Also, data throughput in TP-TDMA is higher than EDCF because of the dynamic utilization of lower bit rate video regions. This shows an admission control capability of TP-TDMA to manage the downlink/uplink data ratio. The overall throughput achieved by TP-TDMA is 28% higher than EDCF even though they are utilizing the same physical layer. However, EDCF has a higher percentage of successfully received voice packets compared to TP-TDMA. The reason is that voice packets in TP-TDMA are not retransmitted due to time-sensitivity.

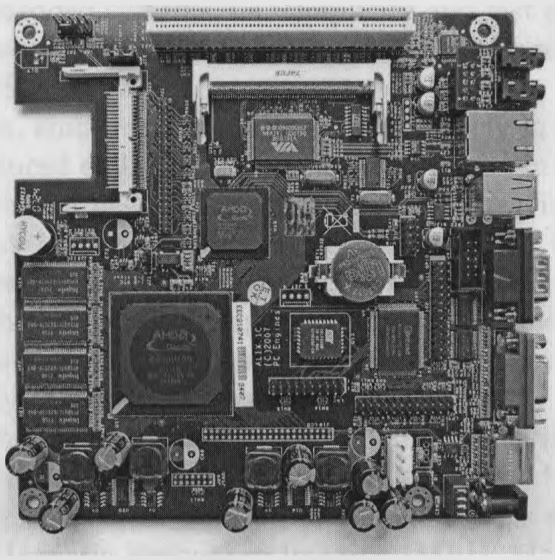

Figure 2.1: Example of an Embedded System

2.2 Embedded Systems

Embedded systems can be defined as the systems that are designed to perform specific and dedicated real-time tasks. The word embedded refers to the fact that those devices are usually built-in within a larger scale system as shown in Figure 2.1. Embedded systems play an important role nowadays in controlling many common devices in our life. Some of the main advantages to using embedded systems in designing a specific application are their flexibility and reliability, as well as the reduction in cost and size. Thus, manufacturers are using embedded systems for mass production and are gaining from their economical advantage. Regardless of the application's size, embedded systems can be part of large scale applications, which benefit from the flexibility they offer. Embedded systems are involved in many aspects of our lives; they can be found in telecommunications systems, personal electronics, transportation system, and in medical equipments.

Modern telecommunication systems utilize embedded systems extensively, e.g., in a cellular network. The Network Switching Subsystem (NSS) deploys a wide range of embedded systems to perform various functions such as identifying caller location. On the end user side, embedded systems are implemented extensively in mobile phones processors. In addition to that, modern networking systems, such as routers and switches, employ different forms of embedded systems to perform scheduling and forwarding tasks.

Personal electronics are considered by far the most consumers of embedded systems as they offer a great deal of efficiency and flexibility to the end users. Many electronic manufacturers nowadays are shifting their designs from being hardware-based to software-based systems. In other words, they tend to utilize the features

that are offered by embedded systems, i.e. simplicity and fast deployment. Examples of personal electronics include: MP3 players, videogame consoles, digital cameras, DVD players, and GPS receivers.

In transportation, embedded systems are used widely in both aviation and automobiles. While advanced embedded systems are involved in the core design of automatic guidance systems of new airplanes, they are also heavily deployed in aviation ground control systems for air traffic control. On the other hand, as the automobile industry shifts towards more hybrid vehicles, embedded systems involvement in the design of those vehicles increases rapidly. Moreover, in most transportation systems, safety-related issues are mainly assigned to software-based systems, which are implemented physically in embedded systems.

In addition to the different uses of embedded systems in various fields, medical technology also benefits from the features, which embedded systems offer; they are involved in a range of medical imaging scanning systems such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI). Moreover, wireless sensor networking utilizes the advances achieved in integrated circuits to implement sophisticated sensors on wireless subsystems, making it possible for companies to create more efficient and responsive systems.

2.2.1 History of Embedded Systems

The history of embedded systems dates back to the early years of single tasked computers, to the 1940-50s, when they were too large and expensive. However, as a result of computer evolution from electromechanical sequencers to the use of integrated circuit technologies, embedded systems have evolved tremendously gaining from the development in transistors technology. As it's always the case in technology; space, defense and military applications were the biggest motivators and also the largest consumers to encourage development in embedded systems.

The Apollo Guidance Computer (AGC) is the first known modern embedded system developed at the MIT Instrumentation Laboratory. At that time, the early 1960s, the AGC took advantage of the new developments achieved in embedded systems to reduce their size and weight; however, the AGC was considered the most risky part of the Apollo project since it involved the usage of new integrated circuit technologies. In 1961, using transistor logic and a hard disk, the D-17 guidance computer for the Minuteman missile was built. Five years later, the D-17 was replaced with a new computer for the Minuteman II missile. The design of this new computer was the first extensive use of integrated circuits and resulted in great cost reductions, allowing the prices of quad NAND gates to drop from $1000/each to $3/each.

The commercial sector was also gaining from the price reductions in the manufacturing of embedded systems, but also it was motivated by the improvements in processing power and the abilities of these systems. Intel introduced its first microprocessor, Intel 4004, in 1971. This microprocessor was simple in its design and was

intended for calculators and other small systems. As the cost of microcontrollers and microprocessors continued to drop, it became feasible to integrate (or replace) some circuitry components, such as variable capacitors, with microprocessors.

2.2.2 Embedded Systems Characteristics

Embedded systems have some distinguished characteristics separating them from general-purpose computers. These characteristics can be listed as follows:

- Embedded systems are designed to do specific tasks rather than being multi-tasked as it's the case of personal computers. And since their nature is to have a limited number of tasks, some embedded systems can be simplified in terms of hardware design costs; in other words, when the performance requirements are low, hardware cost minimization can be achieved.
- Embedded systems are usually implemented in a bigger system to serve a more general purpose; they are not always standalone devices.
- Instructions for embedded systems are stored in read-only memories or flash memory chips; these sets of instructions are referred to as firmware.

The user interfaces of embedded systems vary depending on how sophisticated the system is. Therefore, while single-task systems have no user interface at all, other embedded systems can provide users with graphical user interfaces. Furthermore, complex systems have advanced interface screens allowing the users to interact with them by touch sensing. In contrast, some systems allow modifications in their behavior to be achieved remotely through serial or network connections, i.e. through RS-232, USB or Ethernet cable. Typically, this approach is preferred when the user requires authentications to adjust the functionalities of the targeted system.

Embedded systems can be categorized as microprocessors and microcontrollers. The main difference between the two categories is that microcontrollers tend to have more peripherals on the chip while microprocessors have only the Central Processing Unit (CPU) on the chip. Obviously, the advantage of having more peripherals in microcontrollers' design is to reduce the cost and size. In both, microcontrollers and microprocessors, basic CPU architectures are used, such as the Harvard architecture.

System-on-chip (SoC) is a common configuration of embedded systems in which a complete system of processors, multipliers, and interfaces are located on a single chip. This configuration can be implemented on an Application-specific Integrated Circuit (ASIC) or on a Field Programmable Gate Array (FPGA). As for FPGAs, designers use compliers, assemblers, and debuggers to develop embedded systems software. To communicate with the outside world, embedded systems use peripherals such as:

- Serial Communication Interface, such as: RS-232

- Debugging, such as: JTAG
- Universal Serial Bus, USB
- Networks, such as: Ethernet
- Analog to Digital/Digital to Analog: (ADC/DAC)
- Synchronous Serial Communication Interface, such as: SPI
- Multi-Media Cards, such as: Compact Flash

Since embedded systems are deployed in systems that are designed to run for years continuously, their reliability is always set to very high standards. For that, designers usually avoid introducing unreliable mechanical parts in their embedded systems, and they are put in operation only after being tested carefully several times. There are still some reliability concerns when the design involves having embedded systems. One of the concerns is that embedded systems in many cases cannot be shut down or easily accessed for repair; examples include space systems and undersea cables. Another concern is that those systems must be operating efficiently for safety reasons; examples include: aircraft navigation and reactor control. Also, in some cases, having these systems shut down will cost a large amount of money; this result can be seen clearly in stock markets and in automated sales.

Many techniques are being implemented to avoid, or to recover from, software bugs and errors in the hardware such as such the watchdog timer, redundant subsystems, limp modes, trusted computing base, embedded hypervisor, and immunity aware programming. A watchdog timer resets the computer in case of errors; redundant subsystems are used to allow a switch over in case the main system fails; limp modes provide partial software functionality; the trusted computing base ensures high security and reliability.

2.2.3 Xilinx Hardware Embedded Systems

As a leading manufacturer of FPGAs, Xilinx continues to introduce innovated designs for embedded systems on FPGAs. The Xilinx design of embedded systems on FPGAs consists of three main objects[17]:

- FPGA hardware design; the hardware design requires, in most cases, having a processor and other FPGA hardware. Common processors used in Xilinx boards are MicroBlaze processor and PowerPC processor; more details about those processors are provided later.
- Software platform for processor system; platforms can be either a standalone platform supporting C language for instance or a third-party operating system, such as Linux.

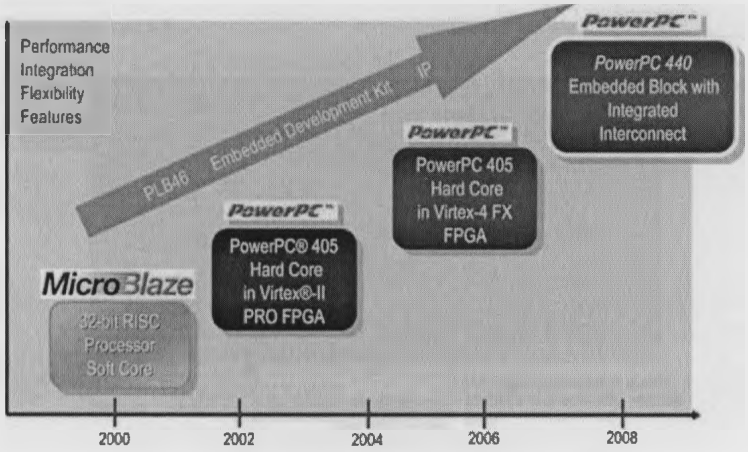

Figure 2.2: Xilinx Embedded Processors Evolution [17]

- User software application in which the user writes an application for specific functions to be performed.

For the hardware design, since 2000 Xilinx boards have allowed development engineers (Figure 2.2) to use two powerful processors: MicroBlaze (soft core) and PowerPC (hard core). MicroBalze was developed by Xilinx while PowerPC is IBM intellectual property; all Xilinx board families support MicroBlaze while only some boards of the Virtex family support PowerPC. In those Virtex boards, which have PowerPC processor integrated into the FPGA, crossbar technology is used to interface between the processor and the outside world.

As shown in Figure 2.3, PowerPC 440 is connected to a Double Data Rate (DDR) memory controller via a separate port called Memory Controller Interface (MCI) to improve system performance and enable access to larger memories. Also, the PowerPC 440 has Direct Memory Access (DMA) ports, which are generally used to connect to the Tri-mode Ethernet MACs (TEMAC). The Processor Local Bus v46 (PLB v46) allows the PowerPC 440 to access any peripherals connected on the bus; while the master bus (MPLB) allows the processor to access peripherals connected to it, the slave bus (SPLB) allows other master buses connected to it to access the MPLB and the MCI memory.

The PLB v46 supports bus widths of up to 128 bits; however, it also supports dynamic bus sizing and programmable burst size. The crossbar technology, shown in Figure 2.4, is built up to allow for a dual five-to-one multiplexer; in other words, one of five masters can be connected to one of the two slaves independently and simultaneously.

The MicroBlaze processor, on the other hand, is a soft-logic processor; and, therefore, it runs in all Xilinx board families. Using block RAMs (BRAM), MicroBlaze can integrate instruction and data caches. On-chip BRAM is accessible via the 32-bit Local Memory Bus (LMB) allowing for low-latency access, while off-chip memories are

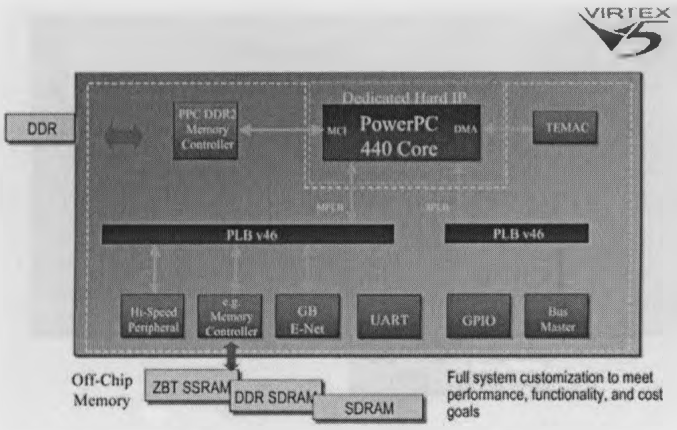

Figure 2.3: IBM PowerPC 440 Processor

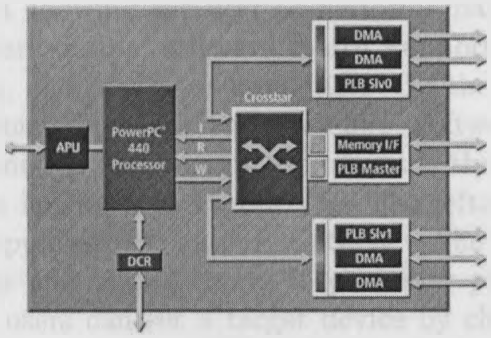

Figure 2.4: Crossbar Technology in PowerPC 440 Processor

accessible via CacheLink to provide high-speed and low-latency access. MicroBlaze utilizes Fast Simplex Link (FSL) channels which are dedicated, unidirectional, and point-to-point FIFO; those channels are 32-bit wide. FSL can provide up to eight input and output interfaces; however, the latest version of Microblaze (7.30a) contains FSL channels that can provide up to sixteen interfaces. MicroBlaze is shown in Figure 2.5.

2.2.4 Xilinx Software Embedded Systems

To support implementing the hardware design, Xilinx introduced a software package called the Embedded Development Kit (EDK) as its boards' software platform. The kit contains all the PowerPC and Microblaze tools and documentation a designer might need. The Xilinx software package consists of two platforms: the Xilinx Platform Studio (XPS) and the Software Development Kit (SDK) [17]. EDK allows hardware development in XPS while software development is allowed in SDK. XPS manages the hardware design of the processor system component and other

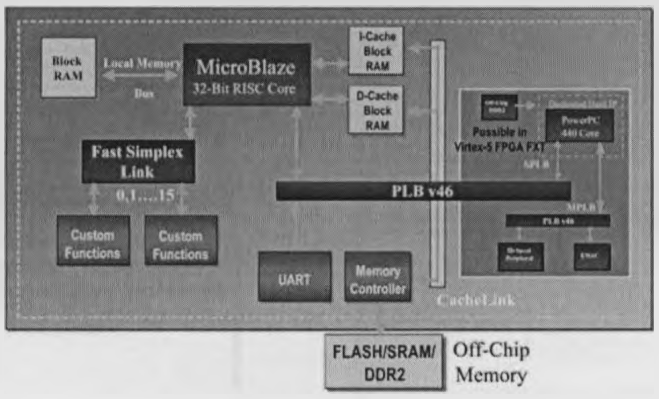

Figure 2.5: Xilinx MicroBlaze Processor

peripherals in the design allowing the user to perform: hardware design, processor netlist generation, software design, software Board Support Package (BSP) generation, software debugging, and hardware simulation. On the other hand, SDK is the software design environment in EDK; and therefore, software application creation, Linker Script creation, and .elf file creation are all made through SDK.

The XPS platform is shown in Figure 2.6. The leftmost side tab consists of three views: Project, Applications, and IP Catalog. The Project view shows the project's current settings and allows access to processor projects files, such as the MHS file. On this tab, users can set a target device by changing the architecture, device size, package, or speed grade. The IP Catalog view enables the designer to add or remove peripherals. On the System Assembly view, the Bus Interface tab allows users to view component datasheets as well as configure them or change their parameters. Ports tabs show all ports connected to a specific component and allow users to reconnect components if needed.

For easier and faster creation of design projects, XPS introduces the Base System Builder (BSB) wizard. Using BSB, the user first has to choose the targeted board from a list of Xilinx predefined boards or link the board definition to files if it was made by a third-party vendor. Then, the BSB will allow the user to configure the system either to be a single-processor or dual-processor system. After that, the processor type, clock frequency, bus clock, and debug interface will be configured. Following that, the user can choose to add or remove peripherals and then configure the Input/Output (I/O) interfaces such as Universal Asynchronous Receiver-Transmitter (UART).

SDK (shown in Figure 2.7) is an application development environment based on Java script and the open-source Integrated Development Environment (IDE) developed by the Eclipse Consortium. It uses a similar compiler as XPS, but it provides clients with more functions and capabilities. Once a hardware design has been created; using XPS, an Extensible Markup Language (XML) file is exported to SDK

Figure 2.6: XPS Platform

to be able to read the design configuration. The design procedure used in SDK to develop a software application for an XPS embedded system is as follows:

- An embedded system is designed using XPS, creating a XML file to be imported by SDK.
- SDK is launched and a workplace is created to read the XML netlist description file.
- SDK creates the board support package and then the software application project.
- In SDK, the user compiles applications for a specific board and then downloads the executable software to the targeted device.
- In the Integrated Software Environment (ISE) Project Navigator, the user will have access to the ELF file on the project.
- Programming file can now be generated.

2.2.5 Xilinx MicroBlaze Processor

The MicroBlaze is a Reduce Instruction Set Computer (RISC) embedded soft core processor that includes the following features [17]:

- 32-bit general purpose registers.

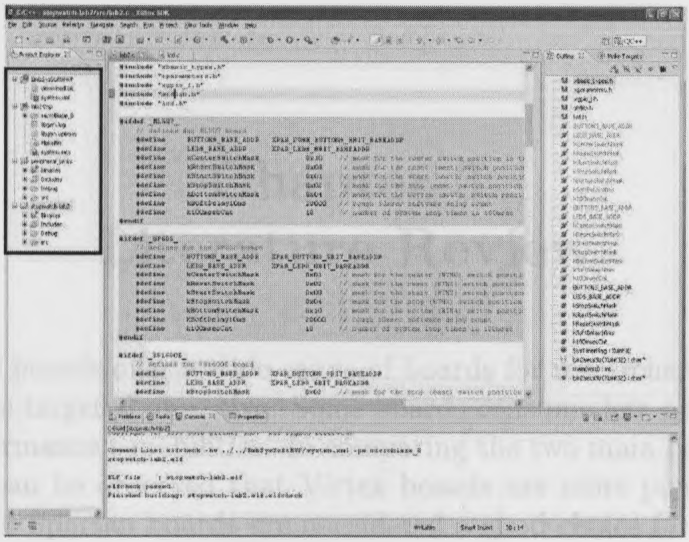

Figure 2.7: SDK Platform

- 30-bit instruction word with three operands and two addressing modes.
- Two 32-bit separate buses; one for instruction and the other one to function as a data bus. Both are compatible with the IBM PLB bus standard and are directly connected to on-chip RAM.
- Single-issue pipeline.
- Instruction and data cache.
- Hardware debug logic, MicroBlaze Debug Mode (MDM).
- Fast Simplex Link support.
- Hardware multiplier.

Instructions in MicroBlaze take one clock cycle; except for the load and store, and multiply which takes two clock cycles. Operating frequency is dependent on the board; for example, Virtex-6 operates at 307 MHz, while Virtex-5 operates at 245 MHz.

Chapter 3
Literature Review

Xilinx families of boards offer a wide range of boards for researchers in order to help them focus on the targeted objective. Some boards consume less power, while others offer higher performance capabilities. In comparing the two main board families that Xilinx offers, it can be observed that Virtex boards are more powerful in terms of performance while Spartan boards are considered a good choice for low total system cost and high-volume applications. More specifically, Virtex boards support a higher number of logic cells and embedded block RAM; while on the other hand, Spartan boards can optimize the targeted design if the performance requirements are not high. For example, Virtex-5 boards support up to 330,000 logic cells and up to 18 Mbits of embedded block RAM, while Spartan-6 boards support up to 150,000 logic cells and up to 4.8 Mbits of embedded block RAM.

This chapter is divided into two sections. The first section is devoted to discussing Virtex-5 LX110T board while the the second section studies MAC layer implementation on boards as well as various MicroBlaze uses.

3.1 Xilinx Virtex-5 LX110T Board

The ML505 evaluation platform hosts a Virtex-5 Field Programmable Gate Array (FPGA) to enable designers to better interact with and explore the board's capabilities. The ML505 board comes with various devices connected to the main FPGA ranging from memory components to Input/output ports such as RS323, Joint Test Action Group (JTAG) port or video output Digital Visual Interface (DVI). This evaluation platform is among the other platforms manufactured by Xilinx in which they share the same printed-circuit board. These boards commonly have a wide range of features or components that makes them useful to test new designs or verify interface compatibilities. A generalized example block diagram of the board is shown in Figure 3.1. In the following sections, we will explore the main components of the XUP5V-LX110T board.

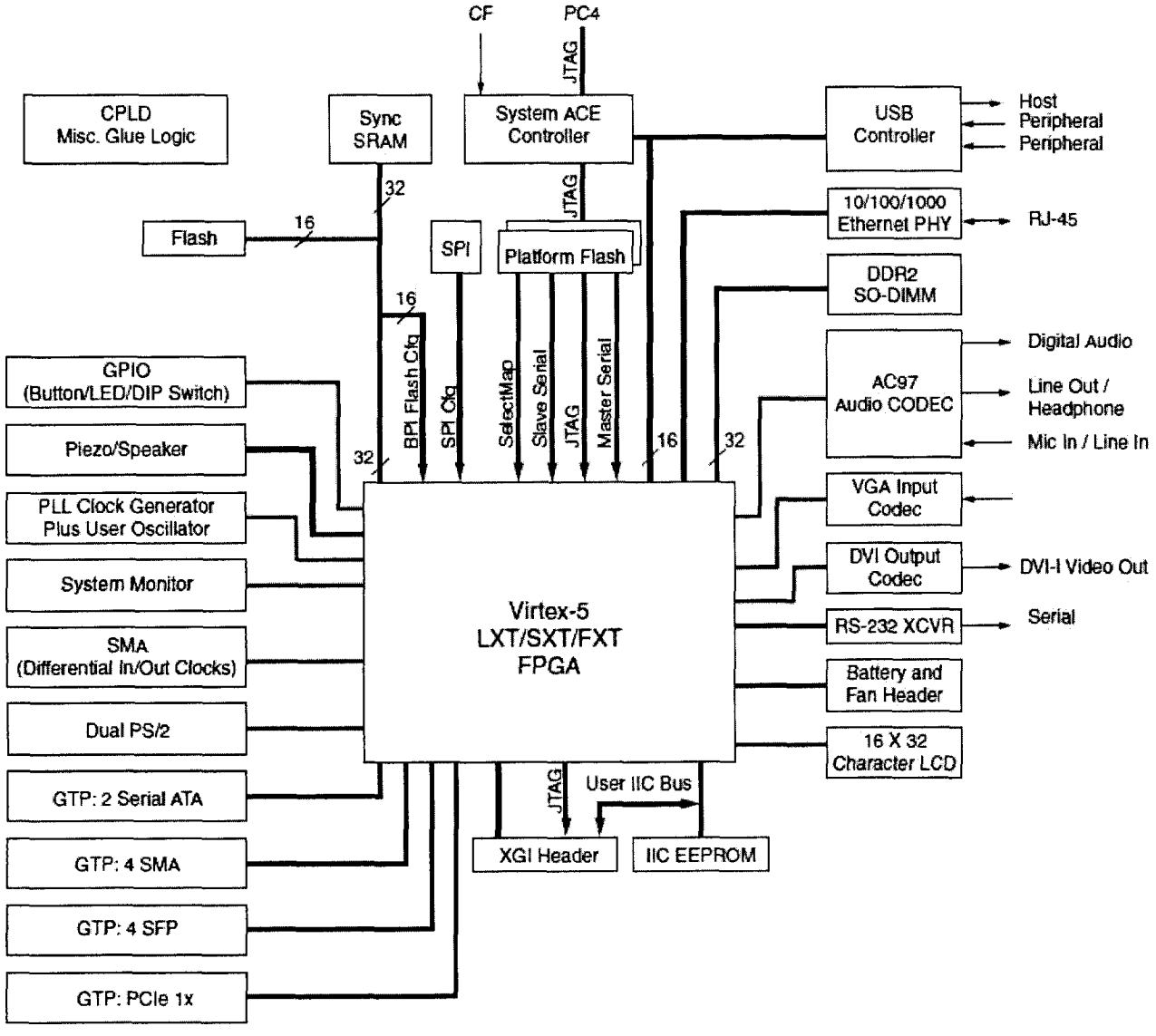

Figure 3.1: Virtex-5 FPGA ML50x Platform[18]

3.1.1 Virtex-5 FPGA

Virtex-5 FPGA is located at the center front side of the board (item number 1 in Figure 3.2), and can be configured in various modes: JTAG, Master Serial, Slave Serial, Master SelectMAP, Slave SelectMAP, Byte-wide Peripheral Interface (BPI) Up, BPI Down, and Serial Peripheral Interface (SPI) modes. To configure the FPGA, one of the following options can be used:

- Xilinx download cable (JTAG)
- System ACE controller (JTAG)
- Two Platform Flash Programmable Read Only Memory (PROM)
- Linear Flash memory
- SPI Flash memory

The JTAG chain starts with the PC4 connector (commonly referred to as J1 pin), followed by the two Platform Flash PROM (U4 and U5), then the Complex Programmable Logic Device (CPLD) (U3), and it goes through the System Advanced Configuration Environment (ACE) controller (U2), and the FPGA (U1). This JTAG chain can be used to program and access the FPGA for hardware and software debugging.The other option to configure the FPGA is the Platform Flash PROM; when using this option, one of two configurations can be selected by using the least significant bits of the configuration address DIP switches. The Platform Flash PROM holds two configuration images; however, four images can be held using compression. The linear Flash memory option supports up to four configuration images, in which data stored in the linear flash is used to program the FPGA (BPI mode). Similarly, the SPI configuration option is achieved through using data stored in the SPI flash memory; however, the DIP switches must be set to 001 to enable this configuration.

3.1.2 DDR2 SODIMM

The DDR2 SODIMM is located on the back side of the board (item number 2 in Figure 3.3). The Memory Interface Generator (MIG) is used to interface the DDR2 SODIMM using the MIG tool. The MIG tool is also responsible for generating the proper FPGA I/O pin selections so that they are compatible with the DDR2 interface. DDR2 memory expansion is achieved by installing the SODIMM modules, which will enable a higher order address.

3.1.3 Differential Clock Input and Output with SMA Connectors

Differential clock signals can be used to derive high-precision signals from the FPGA through 50Ω connectors. An external function generator or clock source can drive the differential clock inputs feeding the global clock input pins of the FPGA. In Figure 3.2, Differential Clock Output is shown as number 3.

3.1.4 Oscillators

The board uses one crystal oscillator socket (X1) as a standard LVTTL-type oscillator. The oscillator socket is powered by a 3.3V supply and a 100-MHz oscillator. The board also comes with a programmable clock generator device that can be used to generate different clocks for the FPGA and its peripherals:

- 27 MHz, 33 MHz, and a differential 200 MHz clock to the FPGA
- 25 MHz to the Ethernet PHY (U16)
- 14 MHz to the audio codec (U22)
- 27 MHz to the USB Controller (U23)
- 33 MHz to the Xilinx System ACE CF (U2)

The crystal oscillator socket (X1) can be seen in Figure 3.2 as number 4.

3.1.5 GPIO DIP Switches

GPIO DIP switches are located in the right-bottom cornor on the front side of the board (item number 6 in Figure 3.2). Eight general-purpose DIP switches are connected to the I/O pins of the FPGA. These DIP switches are an active-high switches.

3.1.6 User and Error LEDs

The total number of LEDs, which are controlled by the FPGA is 15, in which they are active-high. Eight of them are general purpose LEDs situated in a row, five LEDs are located near the North-East-South-West-Center-oriented pushbuttons, and two error LEDs. All the LEDs are colored green except the two error LEDs which are colored red. These LEDs are referred to as number 7 in Figure 3.2.

3.1.7 User Pushbuttons

Located near on the front side of the board (shown as number 8 in Figure 3.2), five active-high pushbuttons serve as general purpose pushbuttons. These pushbuttons are arranged in a North-East-South-West-Center orientation.

3.1.8 CPU Reset Button

In a set of three pushbuttons (shown as number 9 in Figure 3.2), the CPU reset button is the right most button and is an active-low.

3.1.9 RS-232 Serial Port

To allow communication between the FPGA and other devices, the board has one male DB-9 RS-232 serial port (shown as number 12 in Figure 3.2). Since it is wired as a host device, it requires a null modem cable to establish connectivity between the board and the computer (through the serial port). Its maximum communication data rate 115200 bits per second. The board also supports a secondary serial port through the header (J61).

3.1.10 10/100/1000 Tri-Speed Ethernet PHY

The board comes with an Ethernet interface manufactured by Marvell Alaska operating at 10/100/1000 Mb/s. Different interfacing modes are supported by the board: MII, GMII, RGMII (Reduced Gigabit Media Independent Interface), and SGMII (Serial-GMII). It is referred to as number 21 in Figure 3.2.

3.1.11 JTAG Configuration Port

Using the JTAG configuration port (J1), the FPGA can be programmed and debugged. The port supports the standard Xilinx Parallel Cable III, Parallel Cable IV, or Platform USB cable products. The JTAG port is referred to as number 24 in Figure 3.2.

3.1.12 Onboard Power Supplies

The power supply circuitry generates different voltage levels on the board including 0.9V, 1.0V, 1.8V, 2.5V, and 3.3V. For 1.0V, 1.8V, and 3.3V supplies, the Texas Instruments PTH08T2 regulator is used to drive them. The power circuitry is referred to as number 25 in Figure 3.2.

3.1.13 Power, DONE, INIT Indicator LEDs

The Power indicator (PWR Good LED) is on to show that the proper 5V supply has been applied (referred to as number 27 in Figure 3.2. The DONE LED is connected to the DONE pin on the FPGA, and it's on when the FPGA has been successfully configured (referred to as number 28 in Figure 3.2). The INIT LED is on once the FPGA is powered-up and completed its internal power-on process (referred to as number 29 in Figure 3.2).

Table 3.1: Connection and FPGA Pins [18]

Component	Connector on Board	Label	FPGA Pin
Differential Clock	J10	SMA_DIFF_CLK_IN_P	H14
	J11	SMA_DIFF_CLK_IN_N	H15
	J12	SMA_DIFF_CLK_OUT_P	J20
	J13	SMA_DIFF_CLK_OUT_N	J21
Oscillators	X1	USER_CLK	AH15
	U8	CLK_33MHZ_FPGA	AH17
	U8	CLK_27MHZ_FPGA	AG18
	U8	CLK_FPGA_P	L19
	U8	CLK_FPGA_N	K19
GPIO DIP Switches	SW4	GPIO_DIP_SW1	U25
	SW4	GPIO_DIP_SW2	AG27
	SW4	GPIO_DIP_SW3	AF25
	SW4	GPIO_DIP_SW4	AF26
	SW4	GPIO_DIP_SW5	AF27
	SW4	GPIO_DIP_SW6	AE26
	SW4	GPIO_DIP_SW7	AC25
	SW4	GPIO_DIP_SW8	AC24
User and Error LEDs	DS20	LED North	AF13
	DS21	LED East	AG23
	DS22	LED South	AG12
	DS23	LED West	AF23
	DS24	LED Center	E8
	DS17	GPIO LED 0	H18
	DS16	GPIO LED 1	L18
	DS15	GPIO LED 2	G15
	DS14	GPIO LED 3	AD26
	DS13	GPIO LED 4	G16
	DS12	GPIO LED 5	AD25
	DS11	GPIO LED 6	AD24
	DS10	GPIO LED 7	AE24
	DS6	Error 1	F6
	DS6	Error 2	T10
User Pushbuttons	SW10	N (GPIO North)	U8
	SW11	S (GPIO South)	V8
	SW12	E (GPIO East)	AK7
	SW13	W (GPIO West)	AJ7
	SW14	C (GPIO Center)	AJ6
CPU Reset Button	SW7	CPU RESET	E9

3.1.14 Program Switch

This switch is connected to the FPGA's Prog pin. Whenever this switch is pressed, it clears the FPGA's content. The program switch is seen as number 30 in Figure 3.2.

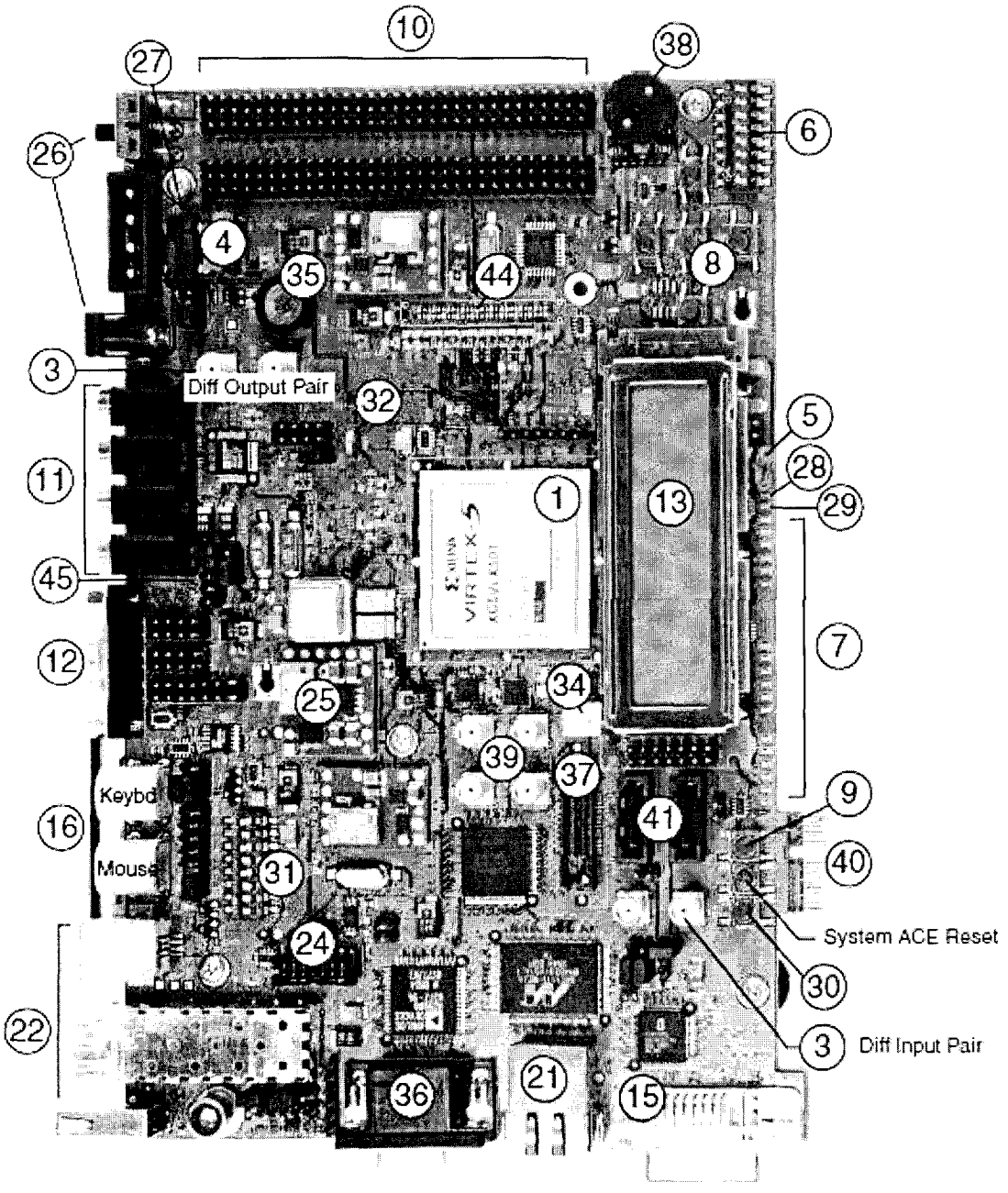

Figure 3.2: ML505 Platform (Front Side)[18]

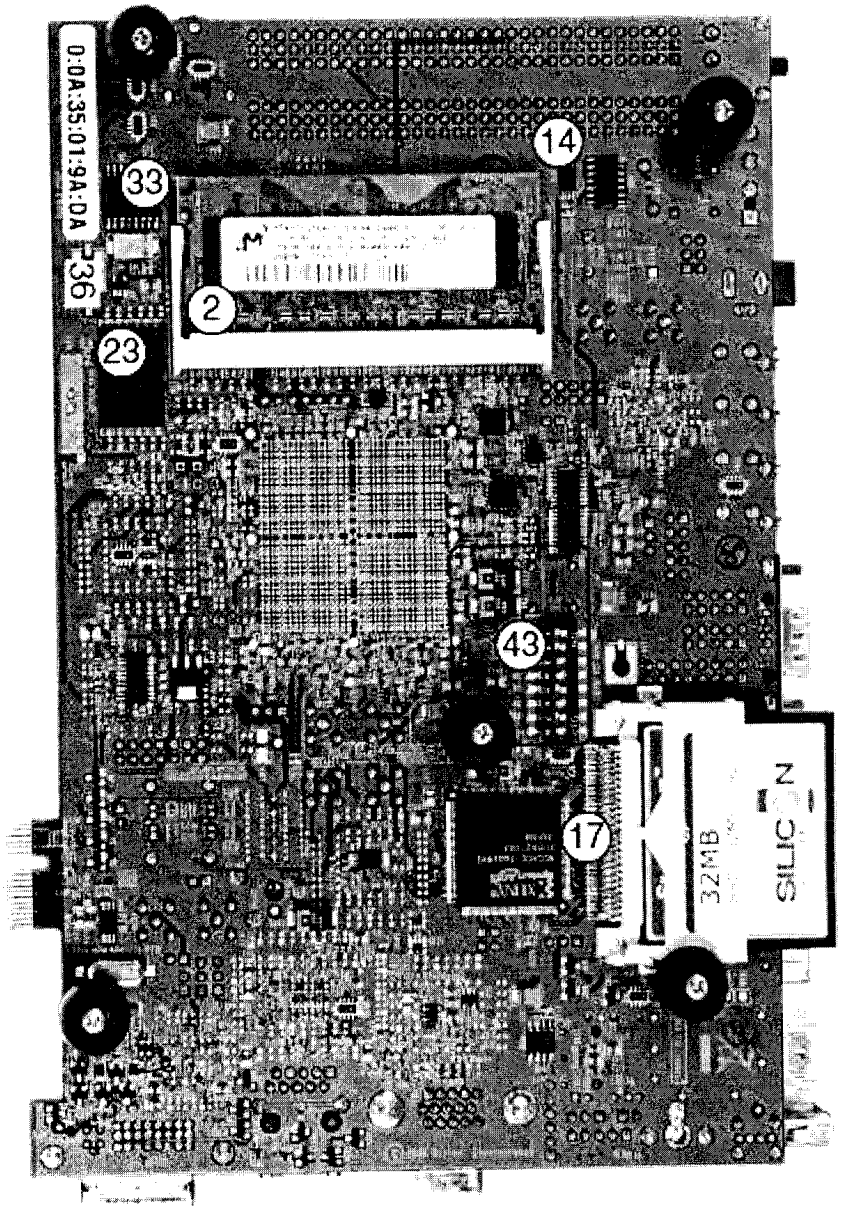

Figure 3.3: ML505 Platform (Back Side)[18]

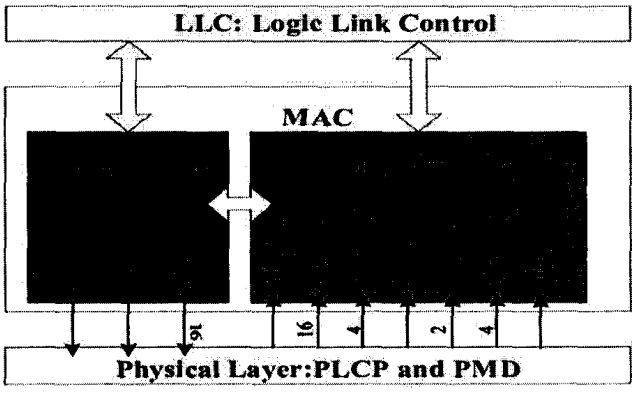

Figure 3.4: PLCP and PMD [19]

3.2 MAC Implementation and MicroBlaze

When it comes to MAC layer implementation, some designers choose to work with FPGA boards as they can provide a more optimal testing environment; however, ASIC boards on the serve as an alternative approach for more customized applications.

3.2.1 Implementation of MAC Layer on FPGAs

Voice over Internet Protocol (VOIP) is one of the most common services that mobile devices are providing nowadays. However, VOIP faces many challenges in which handoff latency between one access point to another is considered to be the main problem. [19] suggests a new solution to reduce handoff latency in IEEE 802.11 by adding some modifications to the mobile devices. The authors suggest two solutions: the first solution is to reduce channel scan time by limiting the devices to pre-informed channels that are known to be the most appropriate. The second solution is limiting the ability of the mobile devices to proceed with handoff only when the future access point has been already selected. With this second approach, handoff latency is minimized since the access point has to allocate needed resources before the connection is established with the mobile device.

The model proposed by [19] consists of two additional sub layers namely: Physical Layer Convergence Procedure (PLCP) and Physical Medium Dependent (PMD). The PLCP acts as the bridge between MAC and the radio transmission, while PMD transmits the bits it receives from the PLCP to the wireless medium. MAC implementation is achieved through a module architecture shown in Figure 3.4 where transmit and receive operations are done separately. Handshaking and interaction signals shown in the figure are described in Table 3.2.

The receiver decodes the PLCP and the PMD into various classes; in other words, it identifies whether the incoming frame is signalization, data, or control so that it can process them accordingly. The MAC receiver component processes probe

Table 3.2: Receiver/Transmitter Component Interface Signals [19]

Signal	Description
Clk	Operation clock
Canal_val	Sets Channel value to physical layer
Order	Used to order the answers in a well defined order
Phy_start	Physical layer notifies receiver start to receive data
Sig_level	Indicates the signal force coming from AP
Prob_data	MT receives the Probe Response frames coming from APs
Phys_data	Data signals from physical layer
Prob_reqst	Notifies transmitter send probe request frame
Ack	Notifies transmitter and acknowledgment frame
Scan_fini	Indicates host that all channels has been scanned completely

response frames as well as controls the Signal to Noise Ratio (SNR) level in order to initialize and operate handoff. On the other hand, the transmitter generates probe requests over different channels for handoff initialization and allows following procedures: event detections, parameters buffering, and message generating.

[20] proposes an evaluation model for a wireless communication systems based on a hardware demonstrator using an FPGA. The main aim of this hardware-based demonstrator is to be as close as possible to wireless systems' behavior in a real environment. The demonstration system consists of an FPGA and a soft processor called LEON3; in which the FPGA realizes the physical layer of orthogonal frequency division multiplexing (OFDM) while the LEON3 supports various MAC protocol evaluations in terms of energy efficiency, throughput, and time synchronization.

Figure 3.5 illustrates the basic structure inside the FPGA; the gray blocks are components imported from external libraries. The critical elements are interconnected via the Advanced High performance Bus (AHB) while low data rate tasks are connected via the Advanced Peripheral Bus (APB). Communication between the PC software and the LEON3 processor is achieved through a bridge between Plugin Bus and the APB.

The MAC layer tasks, such as packets retransmission and framing payload into a packet, are implemented using a LEON3 processor. Using a C language code, these tasks are implemented efficiently. For example, to trigger new attempts to access the medium, a timer is used. As shown in Figure 3.5, the LEON3 processor is composed of all the gray blocks such as the general purpose IO, and the UART. Essentially, all of these blocks act as supporters for the LEON3 CPU block shown in the figure.

One of the problems facing Wireless Local Area Network (WLAN) efficiency is the maximum throughput utilized by the MAC layer in which only 50%, if not less, is actually being utilized. Using Spatial Division Multiplexing (SDM) and two transmit antennas, [21] serves to improve the throughput efficiency by using three FPGAs: two for implementing the physical layer and one for the Enhanced Distributed Channel

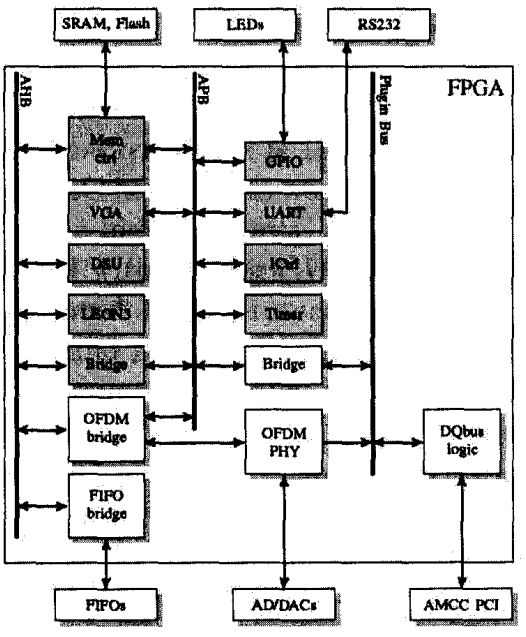

Figure 3.5: Hardware Demonstrator FPGA [20]

Access (EDCA) of the IEEE 802.11e MAC layer. By adding a new block performing acknowledgment, MAC layer efficiency is improved to achieve a throughput more than 100Mbps.

When frames are received, the MAC Receiver module (MRX) checks the header information in order to categorize the incoming frames. Based on the traffic category, all frames are classified into four Access Categories (AC): real-time voice, real time video, best effort, and background work. According to the access categories, various channel access functions are called in to provide differentiated access right for each AC based on the traffic priority. In case a collision occurs, the internal channel collision control function raises the priority for that channel right access. The access right is provided by the transmission opportunity (TXOP), which defines a starting time and a maximum duration for each access in EDCA MAC. The acknowledgment block aggregates several ACK frames into one frame in order to improve the MAC efficiency. Moreover, RTS and CTS frames are used to decrease the probability of channel collision.

The authors of [21] tested their proposed prototype MAC functionality in a point-to-point scenario. They used the following Contention Windows (CW) to maximize the throughput; $CW_{min} = 7$ and $CW_{max} = 15$. The data packet size used was 1828 bytes with 28 bytes as the MAC header and 14 bytes as the ACK; the MAC header rate was 216 Mbps while the ACK rate was 12 Mbps. The improvement achieved on MAC efficiency of the Distributed Coordination Function (DCF) is about 28.5%, i.e., the improved throughput at the receiver is about 61.5 Mbps. Since the authors assumed no packet error in their analysis, the test results are lower than the

theoretical ones for high data rates. The MAC throughput is not only affected by the physical data rate, but also it is affected by the packet error rate for a specific SNR.

The EDCA protocol is investigated to test how much improvement it can add to the MAC efficiency. The packet length is 1926 bytes with a 26 byte MAC header and 14 bytes as ACK; the MAC header rate was 192 Mbps while the ACK rate was 12 Mbps. The throughput of the higher priority AC provides better performance since these packets are transmitted with smaller CW values.

[22] designed an enhanced MAC protocol algorithm to assure sufficient QoS for high-speed wireless data and multimedia applications. The system is built using SDL to implement the main control function; PRISM II chipset and ETRI are used to build the physical layer while the MAC has been implemented using one Xilinx (Virtex board) and one RISC corechip. The enhanced protocol aims to support isochronous and asynchronous traffic in a wireless indoor environment. By adding a subset function, isochronous services are maintained via negotiating the required level of QoS when the connection is established.

In this protocol, the access point plays an important role in reserving the bandwidth for isochronous services using beacon packets which define the required timing and scheduling for those isochronous packets. This allows for dynamic bandwidth allocations to support time-critical applications such VoIP. The authors of [22] decided to move the most time-critical function implementations to the FPGA, using VHDL as the coding method. Their MAC design is based on two processes: transmit process and receive process, in which each is implemented through a number of functions to manage outgoing and incoming traffic.

Considering efficient memory access, the microprocessor's performance, and interface selection, the MAC prototype board is designed. Using a Synchronous Static Random-access Memory (SSRAM) and an ARM7DMI core, one cycle access operation and on-chip memory are achieved as well as high speed data processing with low power consumption. This MAC prototype has many features; they can be listed as: IEEE802.11 compatibility, RISC processor for MAC control functionality, vast SRAM capacity, support of two physical layers (Direct-sequence Spread Spectrum (DSSS) and OFDM), and embedded hardwired MAC logic.

The MAC processor is implemented using SDL, which can be converted to C or C++ language. With the help of a C compiler, an executable program is built using a runtime function library, which contains SDL constructs on the target device, scheduling functions, and implementation operations. To build the Wired Equivalent Protection (WEP) security algorithm, the authors call in external C functions and use hardwired logic in VHDL.

[23] introduced a prototype of Software Defined Radio (SDR), which supports Japanese PHS and IEEE 802.11 wireless standards. In their work, the authors show the hardware and software architecture of the prototype, which consists of a CPU, a Digital Signal Processing (DSP), and an FPGA. The design is portioned so that each component is assigned to specific tasks; in other words, the CPU executes the

MAC functions, while the DPS and the FPGA perform physical layer tasks. The MAC layer is examined by analyzing the resulting throughput in order to evaluate the proposed prototype. The measurement method used was to establish one-to-one cable connection between the AP and the station.

In [24], the authors modeled a Wi-Fi transmitter's MAC layer on VHDL. The design consisted of five main blocks: the Data Unit Interface block, the Controller block, the Payload Data Storage block, the MAC Header Register block, and the Data Processing block. In their work, the authors focused on two blocks: the Payload Data Storage and the Data Processing blocks.

The Payload Data Storage block is divided into two modules: the FIFO module and the Data length counter module. The FIFO acts as a synchronization module to ensure that the rate of entering data matches the rate in the system. The Data length counter module, as its name indicates, simply acts as a counter in order to ensure that a specific number of bits of the incoming data is transmitted. The Data Processing block is divided into three modules: Serializer, HEC, and CRC modules. The Serializer module simply converts parallel input data into serial output data. The HEC module produces the Head Error Check bits then transmits with the PLCP header. The CRC produces the Cyclic Redundancy Check 32-bit field, which helps in error free transmission.

To design a MAC layer for the Ultra-Wideband (UWB) in an FPGA, [25] proposes a MAC controller that consists of three units: a Processor Interface unit, a Control and Data unit, and a Management unit. The Processor Interface unit sends suitable signals to the Control and Data unit after decoding the received commands from the central processor. As its name suggests, the Control and Data unit is responsible for controlling signal lines, which are connected to the physical layer, as well as validating received data. The Management unit read/writes to the data lines in order to manage the physical layer. The MAC controller is also modeled as a Finite State Machine (FSM) of seven different states as shown in Figure 3.6.

3.2.2 Implementation of MAC on Other Boards

Application-Specific Integrated Circuits (ASIC) seem to not offer the most efficient approach to implementing various wireless MAC layer protocols on consumers handsets. Actually, using ASICs for such implementation would be another cost addition on the already expensive approach. Moreover, integrating different standards on a single handset using ASIC will lead to having a separate implementation for each standard. As the demand on advanced wireless devices grows, most manufacturers will tend to introduce devices capable of handling various wireless standards. However, the consumer handset market has very strict requirements; such as being cheap, flexible, and power-efficient, making the implementation of those handsets even more complicated.

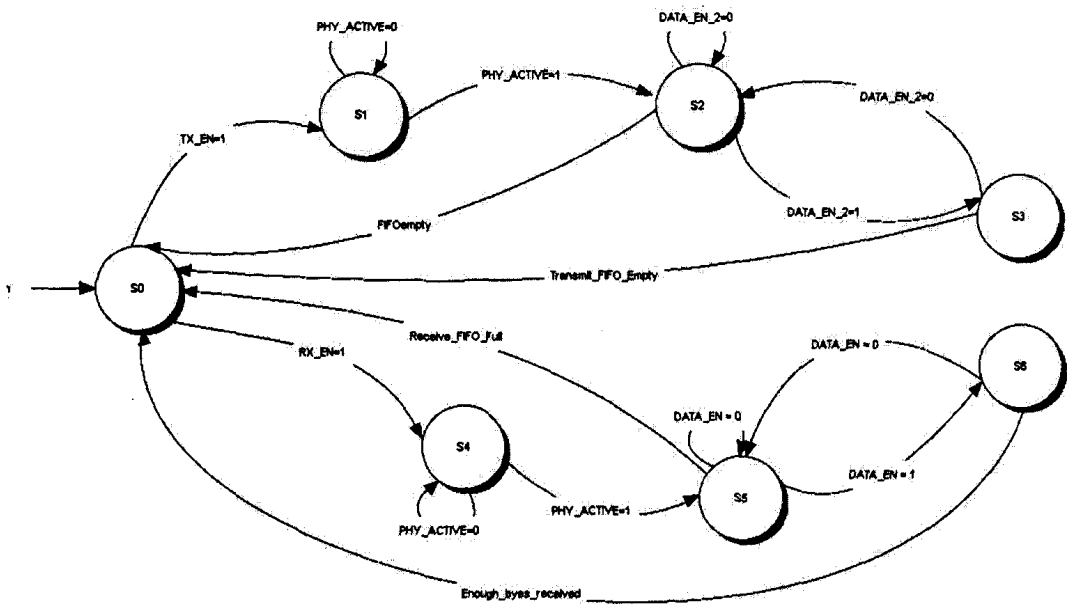

Figure 3.6: WiMedia UWB [25]

This complexity problem is, in fact, applied to both MAC and physical layer standards. For that, [26] proposes a Dynamically Reconfigurable MAC Processor (DRMP) aiming to cope with all MAC protocols commonly using wireless protocols. The authors of [26] investigated three wireless standards: IEEE 802.11 (WiFi), IEEE 802.16 (WiMAX), and IEEE 802.15.3 (WPAN). In their study, they found similarities between these standards in the structure and functionality including header and frame redundancy, and fragmentation. On the other hand, some differences in these standards were found including differences in Automatic Repeat Request (ARR) and RTS/CTS Handshake.

The idea behind the DRMP design is to partition the system into reconfigurable hardware and a general-purpose microprocessor. The reconfigurable hardware part of the system is basically restricted to time-critical functions such as packet processing operations involving transmission and reception. Most of these functions overlap with the three wireless standards considered, motivating faster performance as well as quicker implementation. Conversely, the microprocessor is confined to MAC management and high-level control functions because these functions are not time-critical and are often better done in a software environment.

The MAC hardware has a critical subset that is active as long as the device is in operation mode. Timers, logic for detecting and reacting to events, and state-information storage and update functions are integrated into this critical subset. The DRMP dataflow module is designed to be able to provide a certain sequence of functions by using Reconfigurable Function Units (RFU) connected in different ways to achieve that. The main advantage of designing this dataflow in such a way is that it

is scalable and requires less reconfiguration of data.

The DRMP implements the MAC layer using a function-specific RFUs which reuse hardware resources leading to better efficiency than FPGA or software. Moreover, RFUs reduce the number of interconnections which will yield an improvement over FPGA implementation.

ZigBee is a standard that addresses remote sensing and monitoring applications. [27] discusses a SoC implementation of ZigBee for MAC and the physical layer standard. The MAC layer is managed through an entity called the MLME that is responsible for objects related to the MAC; similarly, the physical layer is managed through an entity called the PLME, which manages objects related to the physical layer.

Two types of channel access mechanisms are allowed depending on the network configuration in which the superframe structure is used. When beacon messages are not enabled in the network, an unslotted CSMA-CA is used. The device will wait for a random backoff time before transmitting its data. If the channel is busy, the device will have to wait for another backoff time; otherwise, it can proceed with the transmission. When beacon messages are enabled in the network, the superframe structure is used. The superframe has one active portion and one inactive portion; transmission is made during the active portion while the device goes into low power mode during the inactive portion. In this configuration, devices are assigned to transmit in either guaranteed time slots or slotted CSMA-CA.

This configuration is implemented using Specification and Description Language (SDL) and the Message Sequence Chart (MSC). The authors used Telelogic's TAU SDL Suite to extract the C code from the SDL file with some modifications to match the code for the 8-bit embedded microprocessor using a 16-bit address instead of the 32-bit. These modifications led to a 70% reduction in the code size, implementing the MAC function with a memory code of 57 Kbytes. The system functionality was confirmed by sending and receiving MAC and physical primitives in the correct order.

3.2.2.1 Weighted Round Robin Scheduler

[28] illustrates how to design a distributed MAC Weighted Round Robin (WRR) scheduler imitating a network layer scheduler. WRR is defined as the simplest approximation of generalization processor sharing for a packet-based network. Each service share has a representative integer weight; for a specific session, the frame is calculated ahead of time by a server that utilizes the service share weights. Using this model, throughput performance can be estimated for a given station's CW size as well as calculating the CW sizes for specific throughput.

The WRR scheduler is implemented in the network layer of the access point of a WLAN; however, the design is considered a MAC layer scheduler instead. Two reasons explain the situation: first, from an uplink prospective, assuming no polling protocol is used, EDCF MAC is the only method to prioritize access to various stations.

Table 3.3: Throughput Performance of Network and MAC WRR [28]

	Network Layer WRR + DCF	MAC Layer WRR + EDCF
Throughput	0.78145 ± 0.00001	0.78875 ± 0.00003
BW1/BW1	1.0000	1.0000
BW2/BW1	0.4998	0.5002
BW3/BW1	0.2499	0.2479
BW4/BW1	0.1250	0.1238
BW5/BW1	0.0625	0.0619
BW6/BW1	0.0312	0.0307

Table 3.4: Access Delay Performance of Network and MAC WRR [28]

	Flow ID	Ave (ms)	Std	95% CI	MIN (ms)	MAX (ms)
Layer 3 WRR	1	22.8	41.4	0.114	7.41	254
	2	37.9	87.3	0.34	15	377
	3	68.1	140	0.768	15	438
	4	129	196	1.53	15	469
	5	250	234	2.58	15	484
	6	491	0.268	0.00417	483	492
MAC WRR	1	15	7.98	0.022	7.37	74.6
	2	29.9	19.2	0.0748	7.37	606
	3	60.7	45.4	0.251	7.37	1180
	4	121	97.1	0.76	7.37	2280
	5	224	188	2	7.37	5750
	6	454	390	5.91	7.37	7910

Second, from a downlink prospective, higher throughput and smaller average access delays can be achieved with a MAC layer scheduler.

When comparing the throughput performances of MAC and network layers, [28] clarifies the fact that MAC's can achieve a higher throughput since the time wasted in backoff is reduced as a result of the competition between multiple access instances. On the other hand, the delay of the MAC layer in WRR of low priority flows is higher than what the network WRR achieved. The reason for that is the need to retransmit more frequently as a result of the virtual resolution policy. Table 3.3 and Table 3.4 show more results.

3.2.3 MicroBlaze Implementation

MicroBlaze is a 32-bit embedded soft processor built by Xilinx and implemented in their FPGA boards. It has a wide range of various uses; some developers use this processor as a scheduler for different network protocols, while others use it a as

queuing system for various processes. In the literature, we found various applications for MicroBlaze including a Real Time Operating Systems (RTOS) scheduler, a Self Adaptive Networked computing Element (SANE) test bench, an embedded Global Positioning System (GPS) receiver, and a video streaming server. On the other hand, some researchers presented special modifications to the MicroBlaze processor to achieve specific targets. For example, [29] proposes a modified architecture of the Fast Simplex Link (FSL) bus to reduce the system size and improve the bus speed by 33%.

To investigate the advantages and disadvantages of different RTOS schedulers, [30] discusses three approaches to scheduler implementations. The first approach is called software-based (SoRTS) in which the implementation is built around a processor that runs the scheduler and other application tasks. The second approach is called software-software (Co-SoRTS) in which two processors co-exist; one processor for the scheduler and the other one for the application tasks. The third approach is called hardware-software (HaRTS) in which the processor runs the application tasks while the scheduler is built-in and hardware-based. To build these approaches, the authors used a Xilinx Virtex-II FPGA board, Xilinx EDK, and ModelSim. MicroBlaze was used to validate the implementation of the software scheduler, with a 50 MHz operating frequency for prototyping purposes.

The first approach, SoRTS, is composed of six elements: MicroBlaze processor, Block RAM memory, On-chip Peripheral Bus (OPB), a communication interface, interrupt and time control, and UART. The application tasks that are related to deadlines, task ID, period, or execution time are handled by MicroBlaze. The Block RAM is used to store tasks scheduled to be executed and tasks that are waiting to be executed in two different data structures; the first is called the Ready queue and the other one is called the Idle queue. While the OPB allows communication between the components using its 32-bit bus, the communication interface allows the software to communicate with the hardware. In other words, by using the communication interface, the MicroBlaze's RTOS and tasks applications can communicate with the interrupt and time control hardware. The UART provides a communication channel between the Xilinx board and the host computer.

The second approach, Co-SoRTS, is similar to the SoRTS architecture but has two MicroBlaze processors instead of one. The tasks which are stored in the Block RAM are performed by one processor while the other processor is used as a RTOS scheduler. In the third approach, HaRTS, employs a dedicated hardware component for scheduling tasks instead of using a software approach. This dedicated component consists of four main modules including a scheduler module, queue control, a communication interface, and time control. The scheduler module is built in three blocks: fail process, running process, and ready process. Using stored parameters in queue control, the scheduler module prioritize the tasks to be performed. The time management is handled by a time control module.

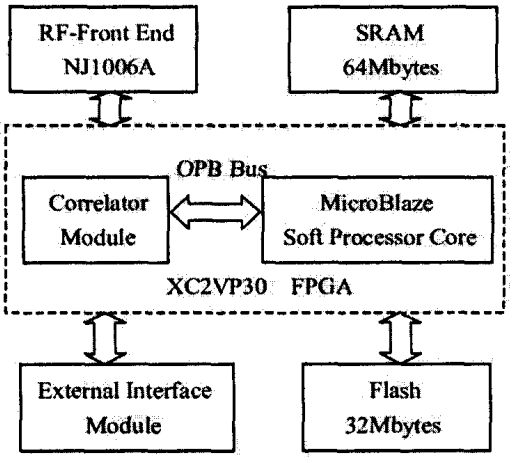

Figure 3.7: Embedded GPS Receiver [32]

In comparison, the author found that CPU utilization in the first approach, SoRTS, is affected by the increase in context switches time while the other two approaches, Co-SoRTS and HaRTS, are not affected. In terms of register usages, HaRTS required the highest number of registers in its communication interface, which was 16 registers compared to two registers by SoRTS and three registers in their communication interfaces.

[31] uses MicroBlaze to build a platform to investigate the SANE concept by using simple CPUs controlling the functionality of data processing blocks. The platform employs a MicroBlaze CPU along with common peripherals such as a DDR RAM controller and a RS232 interface. It also has a simple PicoBlaze CPU, which is used to schedule hardware path operations. This platform consists of three programming levels; the first level is done in MicroBlaze, second level is done in PicoBlaze, and the third is in the hardware.

To design a full GPS receiver using a single chip, [32] necessitates a system that is based on an FPGA hardware environment and MicroBlaze as the system's software environment. As shown in Figure 3.7, the system consists of five main blocks. The RF chip acts as a down converter of the RF signal received to an IF digital 2-bit signal. The signal correlation is done by the hardware on the board; while MicroBlaze is responsible for the baseband signal processing, the Coarse/Acquisition (C/A) code generation and the Digitally Controlled Oscillator (DCO) increment. The SRAM is used to store temporary data, while the Flash chip stores the program. For communication with other devices, the external port module includes serial and debugging ports.

Aside from its powerful computational capabilities, the authors decided to implement their solution using this configuration, FPGA and MicroBlaze, since it will provide more flexibility in accommodating other positioning systems using about the same hardware. In other words, by updating and applying slight modifications to the

software, a hybrid GPS/Galileo receiver can be achieved with dual frequency capabilities. Moreover, the FPGA approach could be used as a test bench for an ASIC design in the future.

The MicroBlaze processor functions in the GPS receiver are mainly; first, interfacing with the custom user-defined systems. Second, controlling and managing the acquisition and tracking phase increment. Third, computing navigation solutions, sending positions, and velocity and time information. The custom user-defined system refers to the correlator which was designed with the help of the Xilinx Tools Generator. Some algorithms are implemented on MicroBlaze for managing and calculating the required navigation solutions.

[33] proposes a platform that enables the video streaming of a pipelined H.263 encoded compression on an embedded FPGA board. Several proposed architectures have been suggested by designers in which they range from hardware-based to systems based fully on a processor. The MicroBlaze processor is used in this system to control the data flow and to instruct the writing/reading from memory.

Chapter 4
System Architecture

Indoor wireless networks, in most cases, are nothing but an extension of the corporate main Ethernet network. In other words, unreachable offices by the wired network are commonly connected to the main network through a wireless hub via an Ethernet port. Since the 90s, Ethernet dominated the wired LAN market by far, ahead of other technologies such as token ring. Therefore, designing a wireless MAC controller should take into consideration the Ethernet's frame structure. However, implementing the MAC controller might require slight modifications on the standard Ethernet frame structure.

The first section of this chapter uses MicroBlaze soft processor to test the board, including its interfaces and memories. The second section introduces a strict scheduler with detailed designs of each component including the system's memory block. This scheduler utilizes the type segment of the Ethernet frame for a different purpose than its original. In fact, the type segment is carrying the class of the payload's traffic; i.e., whether its voice, video, or best effort. The last section proposes a scheduler design with the help of Xilinx Ethernet controller; this scheduler actually implement an FSM model to prioritize the traffic based on its class.

4.1 Board Testing

To test the board, MicroBlaze soft processor offers a convenient approach to verify the board's interfaces [34]. In addition to the memory components needed for the processor, it also has two Input/Output devices which are the RS-232 interface, and the 8-bit LED as shown in Figure 4.1.

As it can be seen in the figure, three types of buses are connecting the processor (microblaze) to other components. The upper set of buses are connected to the local memory's controllers through two slave local buses: *dlmb* (data local memory bus) to *dlmb_cntlr* (dlmb controller) and *ilmb* (instruction local memory bus) to *ilmb_cntlr* (ilmb controller). The lower set of buses are connecting the processor to the peripherals through master bus: *mb_plb* (master bus processor local bus). Similar to local memory bus, *mb_ plb* is connected to to the processor via two buses: *DPLB* (Data plb) and *IPLB* (Instruction plb). This bus is connected to both slaves in this desgin: LEDs and RS232 via *xps_gpio* (XPS general purpose input output) and *xps_uartlite*

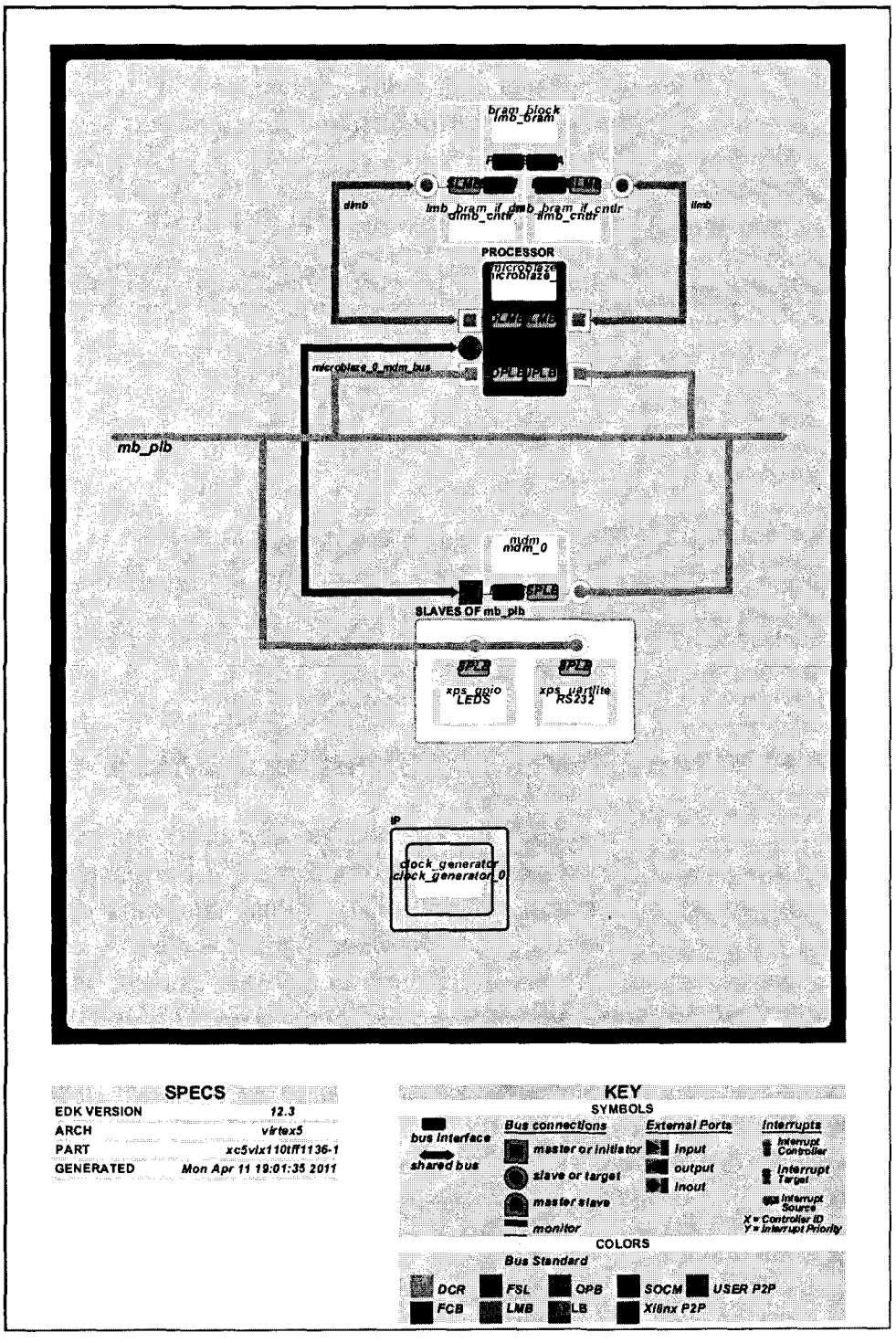

Figure 4.1: MicroBlaze processor used to test the board

(XPS UART lite) controllers, respectively. Finally, to debug the processor, the *mdm* (MicroBlaze Debug Module) is connected to the processor through DEBUG port.

The main aim of this test is to check the communication between the computer and the board through a testing word, and also to examine the internal memories as well as the LEDs. The following (Table 4.1) are the MicroBlaze's system specifications which were automatically generated by XPS wizard:

Table 4.1: MicroBlaze Processor Specifications

Created by Base System Builder Wizard for Xilinx EDK 12.3 Build EDK_MS3.70d	
Target Board	Custom
Family	virtex5
Device	xc5vlx110t
Package	ff1136
Speed Grade	-1
Processor number	1
Processor 1	microblaze_0
System clock frequency	75.0
Debug Interface	On-Chip HW Debug Module

4.2 TP-TDMA Scheduler: Systematic Approach

The systematic approach will be discussed in the coming sections provides a deeper sight at the suggested system compared to the overall design will be suggested in the conceptual approach section. The system design here looks into a single communication setup between the base station and the wireless station regardless of its direction. The system consists of four main components as shown in Figure 4.2: Controller, MEMORY block, Scheduler, and Frame Constructor.

As it can be observed from the system, the Incoming Frame (shown in Figure 4.3) is split into two segments where Frame Spec is sent to the Scheduler and Payload is stored in the MEMORY block. Once the Scheduler decides which Frame Spec to be transmitted, the Frame Constructor calls the associated Payload (using the Payload Tag generated initially by the Controller) and then constructs the Outgoing Frame.

4.2.1 Controller

The Controller (shown in Figure 4.4) consists of three blocks: Preamble Check block, CRC Check block, and the Frame Ripper block. The Preamble Check block performs mainly a comparison between the first 64 bits of the incoming frame with the standard

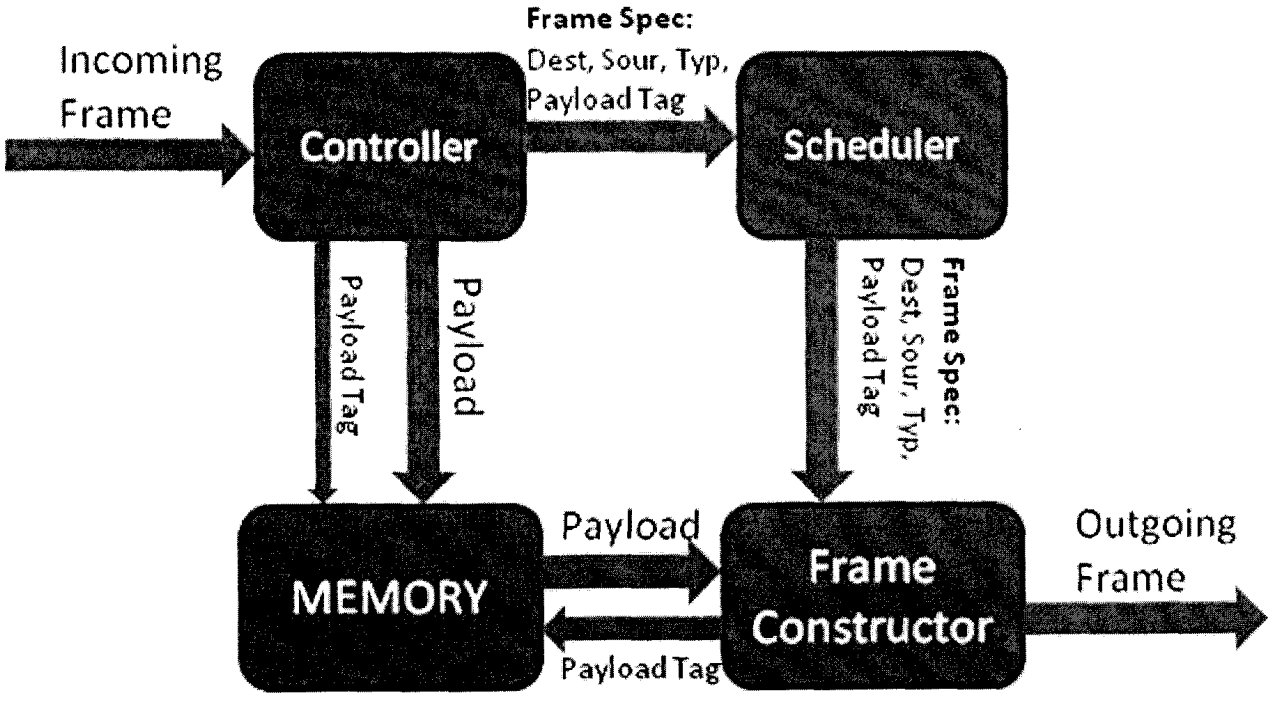

Figure 4.2: System's Main Components

Figure 4.3: Fields of the Incoming Frame

expected preamble of Ethernet frame. This comparison operation is implemented using an XOR gate; if the result of the comparison is all zeros, then the frame preamble is correct. CRC Check block tests the incoming frame's payload to check whether there were errors during the transmission. The CRC Check block contains multiple Wrapper Adders used to compare the CRC field with the payload. If the comparison result was all ones, then the CRC is correct, i.e., no errors during the transmission. At this stage, the incoming frame enters the Frame Ripper block which performs two main functions:

- Fragments the frame to two pieces, Frame Spec and Payload.
- Generates the Payload Tag.

The Frame Spec is actually carrying three fields of the incoming frame in addition to the Payload Tag. Those three fields are: the destination address, the source address, and the type of this frame.

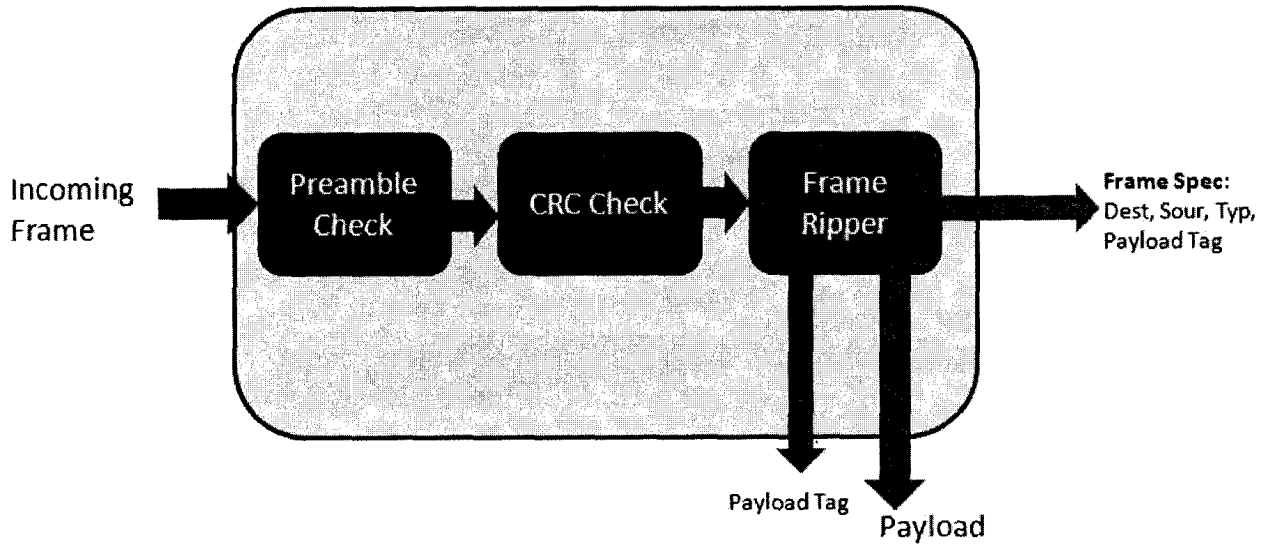

Figure 4.4: Controller Detailed Design

The Frame Spec is sent to the Scheduler while the Payload and the Payload Tag are sent to the MEMORY block. The Payload Tag is generated in order to keep a track of the payload associated with a particular Frame Spec; in addition to that, the Payload Tag is used as an address in the MEMORY block as well as a method to fetch the desired Payload from the MEMORY by the Frame Constructor.

4.2.2 MEMORY Block

The MEMORY block acts as a storage element to store the payload meanwhile scheduling processes are being performed. This block, as shown in Figure 4.5, consists

of a Defragmenter, multiple RAMs, and a Fragment. The Payload and the Payload Tag are sent by the Controller to the MEMORY block, where the Payload Tag is used to create the RAM address in which the associated Payload will be stored.

The Defragmenter works as a splitter of the payload received from the Controller in which segments of 128 bits are sent to each RAM block with a copy of the Payload Tag to store that segment in a specific address. On the other hand, the Fragment receives the Payload Tag to be read from the Frame Constructor, and then calls it from the RAM blocks to gather it in one segment to be sent to the Frame Constructor. It is to be noted that payload segments are stored in the same RAM address in each RAM block.

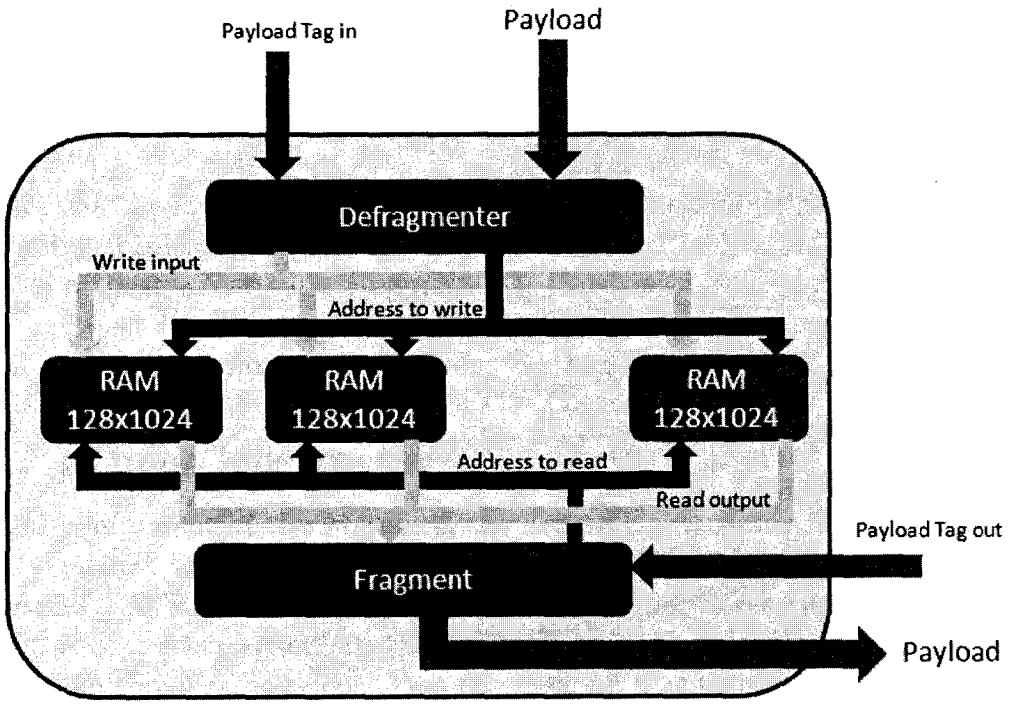

Figure 4.5: MEMORY Block

4.2.3 Scheduler

The Scheduler, shown in Figure 4.6, consists of two main blocks; a Classifier and a Prioritize. The Classifier simply distinguishes different Frame Spec classes of traffic and then sends each Frame Spec to the Prioritize according to its class. Basically, the Classifier checks the first two bytes of the Typ signal; 0x0011 refers to a voice payload, 0x0022 refers to a video payload, and 0x0033 refers to best effort payload.

The Prioritize simply works as a strict scheduler in which only four segments are examined in order to sort them. Priority is always for voice payloads; once there are no voice payloads to schedule, video payloads are set as a second priority leaving

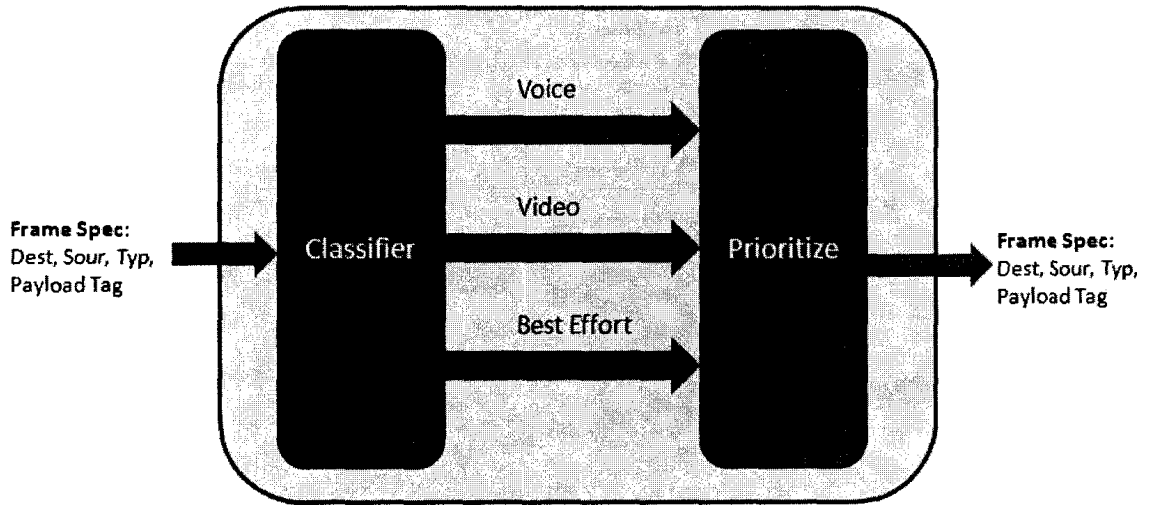

Figure 4.6: Scheduler for Straight Scheduler Approach

best effort as the lowest priority. To explain how the Prioritize works, let's consider an incoming frame of 4 segments. The first segment is of a video type (segment A), second is of best effort type (segment B), third is of voice type (segment C), and fourth is of best effort (segment D). The resulted outgoing frame should be sorted as follows: segment C, segment A, segment B, then segment D.

4.2.4 Frame Constructor

The final component of the system is the Frame Constructor. As its name suggests, it simply gathers all frame components and then constructs the frame to be ready for transmission. As shown in Figure 4.7, the Frame Spec is received from the Scheduler and then a copy of it is sent to the MEMORY block in order to call the associated payload with that Frame Spec. Once the payload is entering the Frame Constructor, a copy of the payload is sent to the CRC to generate the 32-bit CRC. The Frame Constructor provides Preamble and Frame Gap generation to form the final Ethernet outgoing frame.

4.3 TP-TDMA Scheduler: Conceptual Approach

TP-TDMA protocol can be described through the algorithm shown in Figure 4.8. This protocol is designed for a customized system configuration in which a base station serves four clients. Each client receives its traffic (downloading frames) on segment-base according to the traffic class, while clients are allowed to upload their traffic similarly based on their class.

Voice traffic is assigned fixed-sized slots on both downlink and uplink frames, while video and best effort traffic is allocated their bandwidth according to the chan-

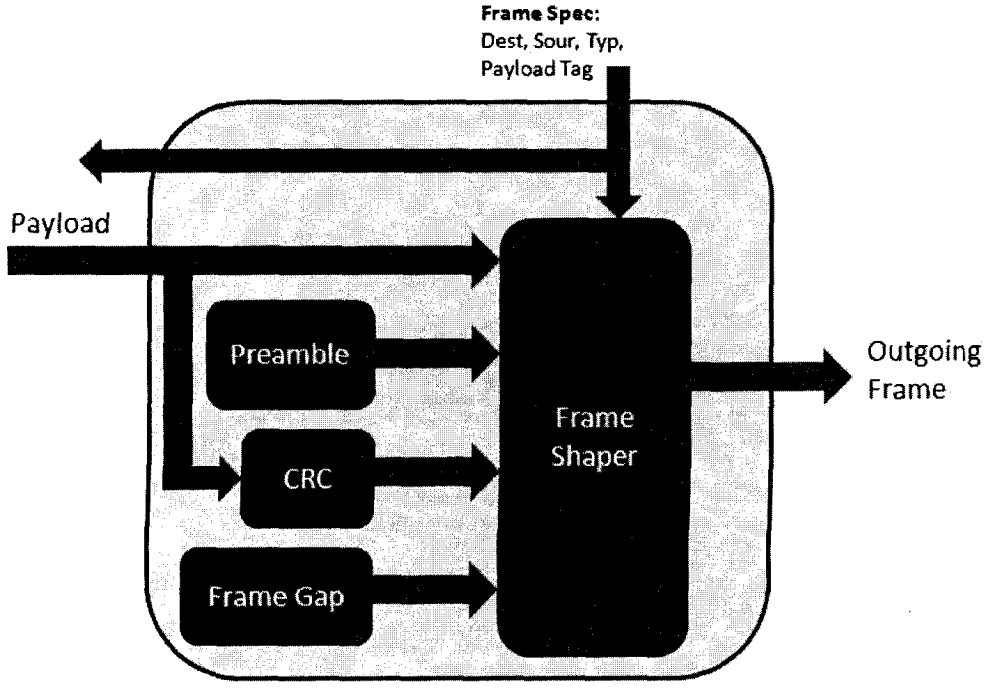

Figure 4.7: Frame Constructor Main Blocks

nel availability. However, video traffic is prioritized over data traffic which will be queued for transmission once video packets are delivered. The downlink frame starts with a beacon packet to carry the scheduling information to all stations in the system. The MAC header contains both transmitter's and receiver's addresses which are distinguished using an addressing system of one byte to identify the station. In that byte, the first four bits are assigned to the network id while the last four bits are assigned to the station id.

To implement the TP-TDMA protocol using Xilinx Vertix-5 FPGA, the system's building blocks consists of three subsystems; an Ethernet Controller, Memory, and a Scheduler as shown in Figure 4.9. The Ethernet Controller will simply receive Ethernet frames and process them so the Scheduler can sort them according to the traffic class. The traffic is fed to the system through a standard RJ45 LAN port, in which the Ethernet Controller will separate the payload from other frame's sections. The payload will be stored in the memory while scheduling processes are taking place in the Scheduler.

4.3.1 System's Components Description

As shown previously in Figure 4.9, the system consists of three components. The Ethernet Controller and the Scheduler will be discussed in further details in the following sub-sections.

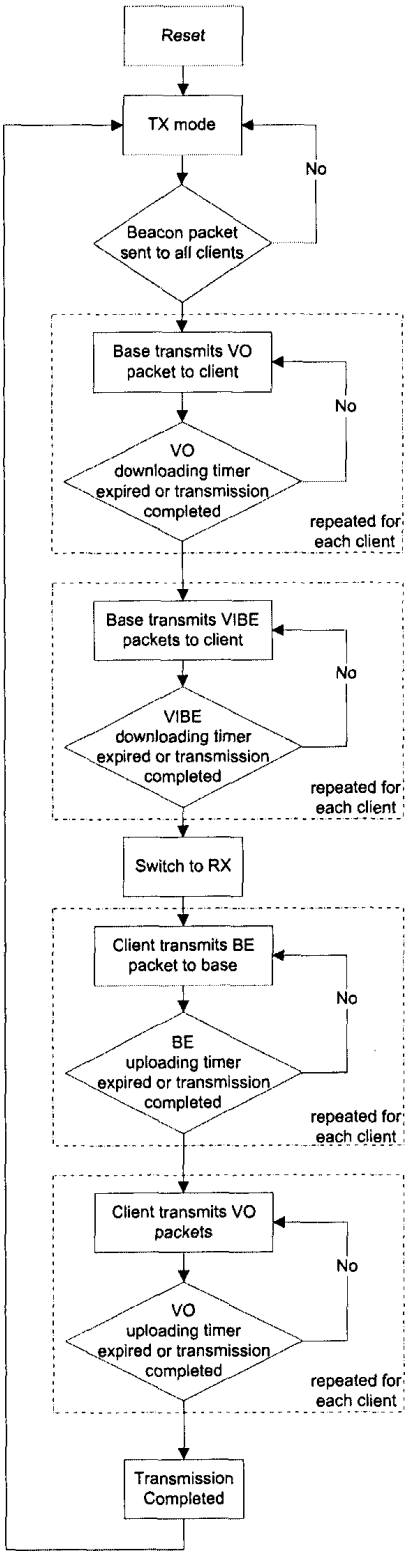

Figure 4.8: TP-TDMA Algorithm

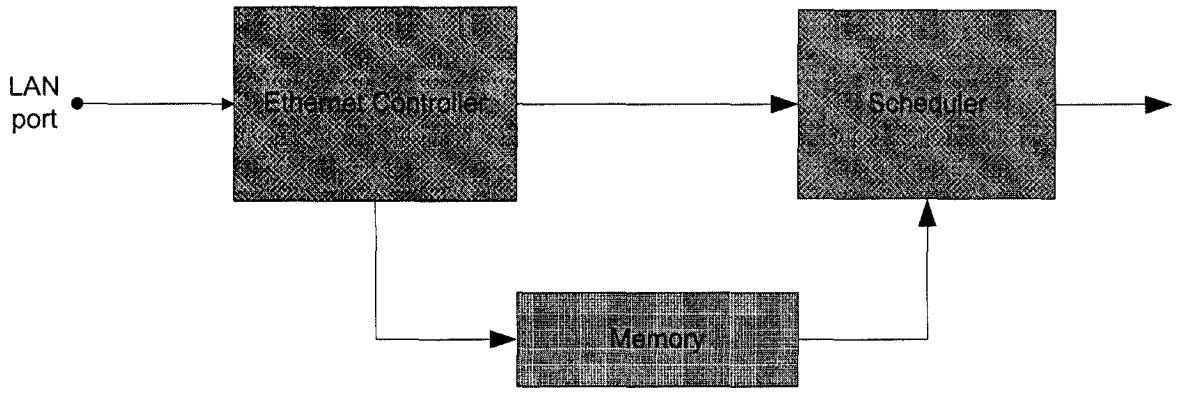

Figure 4.9: System Building Blocks - Ethernet Controller and Scheduler

4.3.1.1 Ethernet Controller

Using ISE CORE Generator, an Ethernet Controller can be created either with the standard features or with customized features to suit the required design. Therefore, to design the Ethernet Controller for TP-TDMA protocol, the CORE Generator creates The LogicCORETM IP Tri-Mode Ethernet Media Access Controller (TEMAC) [35] with the following features. The physical interface type is chosen to be Gigabit Media Independent Interface (GMII) and the MAC speed is Tri speed, i.e. 1 Gbps, 100 Mbps, or 10 Mbps. The client interface is designed to enable clocks at transmitter (TX) and receiver (RX). Management interface is chosen to be accessed with a configuration vector containing all management details. Also, the controller is designed to have full-duplex communication and with no address filter.

As shown in Figure 4.10, the created Ethernet Controller consists of five components. The first two components are TX and RX interfaces acting as the client interface facilitating communication between the upper layer and the Ethernet Controller. Clock/Speed Indicators receive the clock that will control the TX and RX interfaces as well the reset signal to control the whole core. It is to be noted that the TX and RX interfaces can have independent clocks. GMII Physical interface communicates with the physical layer to send frames to the upper layer and receive frames from the physical layer. The Configuration component is an essential component of this core; it has only one signal of 68-bits carrying all management settings [36]. Further details about configuration vector are provided in Appendix A.

Table 4.2 describes all signals of the Ethernet Controller. When there is a frame to be transmitted on *clientemactxd* port, the frame should be validated by having the port *clientemactxvld* and *clientemactxenable* high. After the first byte is transmitted, *emacclienttxack* goes high. When a frame is sent from the MAC to the RX interface, it will be received on *emacclientrxd* port.

On the physical side, frames are received to the MAC through *phyemacrxd* port; while they are transmitted to the physical side from the MAC through *emacphytxd*

Table 4.2: Signal Description of the Ethernet Controller

Signal	Description
clientemactxd (8)	Frame data to be transmitted
clientemactxdvld	Control signal for clientemactxd
clientemactxenable	Clock enable for TX interface
emacclienttxack	Handshaking signal
clientemacrxenable	Clock enable for RX interface
emacclientrxd (8)	Received frame data
emacclientrxdvld	Control signal for emacclientrxd
emacclientrxgoodframe	To indicate good frame reception
emacclientrxbadframe	To indicate bad frame reception
reset	To reset the Ethernet controller
txgmiimiclk	Clock for the transmission of data
rxgmiimiclk	Clock for the reception of data
speedis100	Asserted when the Ethernet is operated at 100 Mbps
speedis10100	Asserted when the Ethernet is operated at 10 Mbps
phyemactxenable	Clock enable for GMII Physical interface
phyemacrxd (8)	Received data from PHY
phyemacrxdv	Control signal for phyemacrxd
emacphytxd (8)	Transmitted data to PHY
emacphytxen	Enable control signal to PHY
emacphytxer	Error control signal to PHY
tieemacconfigvec (68)	Configuration management vector

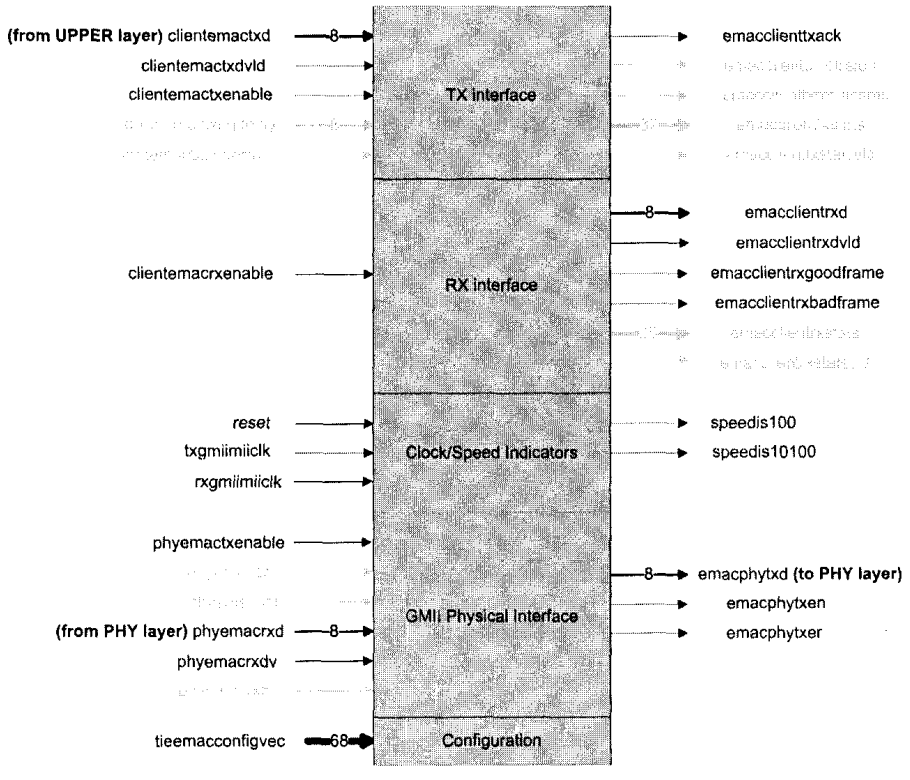

Figure 4.10: Ethernet Controller Components and Signals

port. Frames are transmitted on ports of 8-bits format as mentioned in Table 4.2.

4.3.1.2 Scheduler

The Scheduler is modeled as an FSM of 18 states shown in Figure 4.12. The principle idea of this FSM can be simplified as follows; State00 is the reset state, State01 is the beacon transmission state, State02-State05 are the voice downloading states, State06-State09 are the video and best effort downloading states, State10-State13 are best effort uploading states, State14-State17 are the voice uploading states.

In State00, the system forces reset port to go high and as a result the core resets all ports. Once *time_max* transition signal is high, i.e. the time limit to stay in State00 expires, the system moves to State01 in which it will broadcast a beacon message to all stations. Similarly, once *time_beacon* goes high, the system moves to Stat02. For State02-State05, the transition signal is *time_VO*; it goes high once the station sent its voice frame or the time limit has expired. For State06-State09, the transition signal is *time_VIBE*; it goes high once the station sent its video and best effort frame or the time limit has expired. For State10-State13, the transition signal is *time_uBE*; it goes high once the station uploaded its best effort frame or the time limit has expired. For State14-State17, the transition signal is *time_uVO*; it goes high once the station uploaded its voice frame or the time limit has expired.

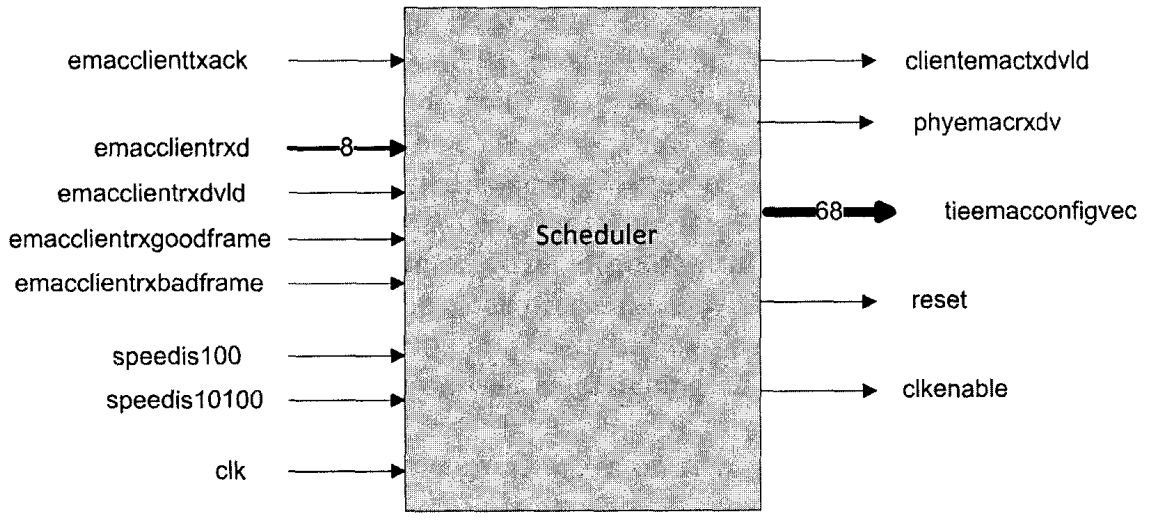

Figure 4.11: Scheduler Block

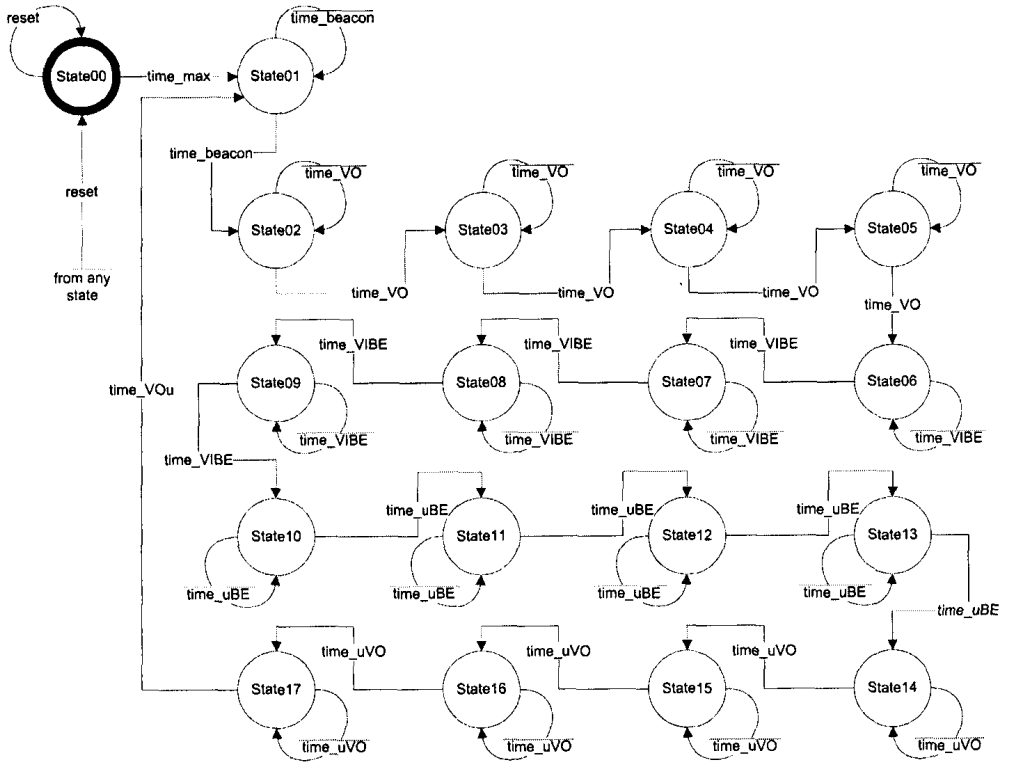

Figure 4.12: FSM Model for the Scheduler

4.3.2 Design Implementation with VHDL

Using VHDL, the system can be implemented by defining the memory and the Ethernet Controller as components added to the Scheduler. Since the memory is acting as an input data feeder, the code will use specific input frames in order to compare them to the resulted output. The code assigns specific time for the transition signals; in addition to that, a timer signal is acting as the checking signal whether the time limit has expired or not[37][38]. Two signals perform the transition from one state to another; *pr_state* and *nx_state*. After declaring all involved signals, two timing processes are defined. One is the clock process, and the other one is the reset process; both of them use *wait for* command. There are two processes that make the transition from one state to another as well as assigning all corresponding signals' values at specific state[39].

The first process is sensitive to changes on the clock and reset signals. In this process, a counter signal (*count*) is assigned the value that had been given to the timer of the current state. Once the timer equals to the count, the process moves to the next state and resets the count. However, if the reset signal is high, the process goes back to State00 regardless of the counter's value; also, it resets the count.

The second process is sensitive to changes on *pr_state*. Once *pr_state* is assigned a new value, i.e. the FSM moves to a different state, the process moves to the next state using case and when commands. All signals are assigned the corresponding values at that state. It can be noticed that *tieemacconfigvec*, *clientemactxd*, and *phyemacrxd* signals are the ones being assigned new values at each state. The signal *tieemacconfigvec* contains the address of the frame's destination (in the last 48 bits), *clientemactxd* carries the frame sent from the upper layer to the MAC layer, while *phyemacrxd* carries the received frames from the physical layer.

Chapter 5
Results

This chapter introduces and discusses the results obtained from the work of Chapter 4. The Results chapter is divided into three sections; similar to the approach followed in the System Architecture chapter. This chapter investigates the methods and approaches proposed to test the targeted board, as well as the scheduler design suggested for triple-play services.

The first section presents the testing results obtained using MicorBlaze soft processor. The aim of this test was to investigate the possibility of establishing communication between the board and the computer as an initial step to utilize MicroBalze's capabilities to program the board. The following sections present the results achieved for the proposed scheduler. In the second section, the systematic approach is examined by testing each component of the design through applying various input signals to the system. Similarly, in section three, the conceptual approach is examined by testing both the Ethernet Controller and the Scheduler by applying various input signals to observe the system's responses.

5.1 Board Testing Results

5.1.1 Results Presentation

Designing a MicroBlaze soft processor is a considerably smooth process thanks to XPS wizard. However, customizing that design to the targeted objective introduces some complications as to figuring out the specifications required to be given for hardware deployment on the board. In fact, the first step to implement the design on the board is to identify the processor's external pins and associate them with the correct pins on the FPGA. That association is achieved by creating the ucf file, which stands for user constraints file. After adding the ucf file to the main design, the processor's creation phase is completed; i.e. the hardware phase of creating the MicorBlaze processor is completed.

In order to program the processor, the design should be exported to SDK which will create the software application controlling the processor. SDK at this stage creates a hardware platform to match the processor's design specifications that will allow C coding files (the software application) to instruct the processor. Once the

software application is checked, the code can be downloaded to the board. To observe the results, a Hyper Terminal session is established with the following specification:

- Bits per second: 115200.
- Data bits: 8.
- Parity: None.
- Stop bits: 1.
- Flow control: None.

The board is now connected to the computer using a serial cable via RS232 port, while the JTAG port is connected to the computer using the downloading cable. This concludes the physical setup on the board making it ready to receive the bit stream file.

The first test is intended to ensure that codes are correctly deployed on the board and then correctly sending associated messages back to the computer to be displayed on the Hyper Terminal. For that, the code simply runs a print function with a hello world message to be displayed as follows:

```
print("Hello World");
```

The second test is intended to check the communication between the FPGA on one side and the peripherals in the processor on the other side. Peripherals in the design include the LED and the RS232, both of them are tested using different functions. The test starts with the following message:

```
print("---Entering main---");
```

Then the function to test peripherals is called and print the following message:

```
print("Running GpioOutputExample() for LEDS...");
```

If the communication is established correctly, the following message is printed on the screen:

```
print("GpioOutputExample PASSED.");
```

If not, the following is printed:

```
print("GpioOutputExample FAILED.");
```

After that, another function is called printing the following message:

```
print("Running UartLiteSelfTestExample() for mdm_0...");
```

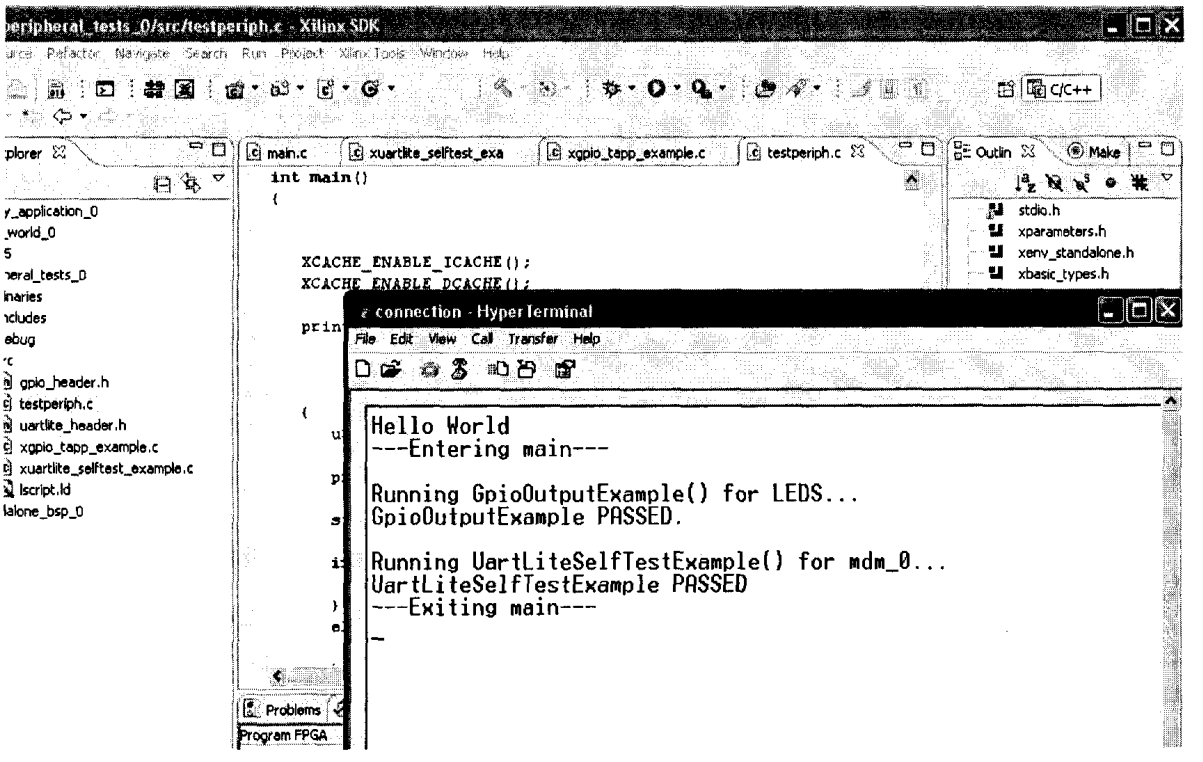

Figure 5.1: Board Testing Result

Similar to the previous the function, this function checks the communication between the board and the computer through the RS232 to ensure that debug function is feasible. If the communication is established correctly, the following message is printed:

```
print("UartLiteSelfTestExample PASSED");
```

If not, the following is printed:

```
print("UartLiteSelfTestExample FAILED");
```

The following message concludes the test:

```
print("---Exiting main---");.
```

Figure 5.1 is a print screen of the test showing the results obtained. As it can been seen, the communication is established correctly and both tests' results were successful. To evaluate the board's utilization this design required, Table 5.1 provides some utilization figures and usages in terms of slice registers and look up tables (LUT). Further details are presented in Section B.1.

Table 5.1: FPGA Usage Report - Device Utilization for MicroBlaze Testing

Slice Logic Utilization	Available	Used (Utilization)
Number of Slice Registers	69,120	1,631 (2%)
Number of Slice LUTs	69,120	1,879 (2%)
Number of occupied Slices	17,280	965 (5%)
Number of bonded IOBs	640	12 (1%)
Number of BlockRAM/FIFO	148	8 (5%)
Number of BUFG/BUFGCTRLs	32	2 (6%)
Number of BSCANs	4	1 (25%)
Number of DSP48Es	64	3 (4%)
Number of PLL_ADVs	6	1 16%)

5.1.2 Results Discussion

The main objective of creating a MicroBlaze soft processor is to test the board's connectivity with the computer and observe the communication results on the Hyper Terminal. The "Hello World" message in Figure 5.1 shows that the code communicates properly with the processor, allowing the second test to focus on testing the peripherals only. Similarly, we can see that the LEDs test as well as the debug connectivity passed. Another observation can be made on the board's utilization made by MicroBlaze as seen in Table 5.1. In that table, it can be easily concluded that MicorBlaze's consumption is relatively low, as only 2% is only consumed of the board's Registers and LUTs.

Even though the MicroBlaze soft processor approach seems remarkably a smooth implementation approach, this soft processor requires a great deal of technical training in order to utilize its capabilities. In addition to that, since MicroBlaze is an intellectual property of Xilinx, it makes it harder for academic researchers to get some of MicroBlaze's features. However, MicroBlaze would still be a highly recommended approach to follow if there were no time frames to meet as it might be a time consuming process to search for available resources other than Xilinx training programs. Another obstacle was observed along with MicroBlaze implementation that in most tutorials Xilinx provides, only Spartan boards were the family of boards used. That created an obstacle for this research as Virtex board is the one targeted for the proposed design. For instance, obtaining the accurate pins' assignments for MicoBlaze on Virtex board was by itself a lengthy search.

5.2 The Systematic Approach Results

5.2.1 Results Presentation

This section examines the performance of each component in the design individually, rather than testing the overall system. As seen in Figure 4.2, the system consists of four subsystems; Controller, MEMORY Block, Scheduler, and Frame Constructor. Each component in the four subsystems will be tested separately by feeding predefined input signals and observe its response.

5.2.1.1 Controller

The first block in the Controller shown in Figure 4.4 is the Preamble Check block. As mentioned earlier, this block simply checks whether the first 64 bits of the frame are matching the standard Ethernet preamble. To test this block, a chain of frames are fed to the system to observe its output. In Figure 5.2, it can be seen that the *preamble_check* port is all zeros indicating that first frame's preamble is correct (when the input frame is x0AAAAAAAAAAAAAAB). On the other hand, the second and the third frames' preambles are not matching the standard Ethernet preamble; therefore, the results on *preamble_check* port are not entirely zeros. Based on that, we can say that this block performs the required function by distinguishing correct preambles to indicate new incoming frames.

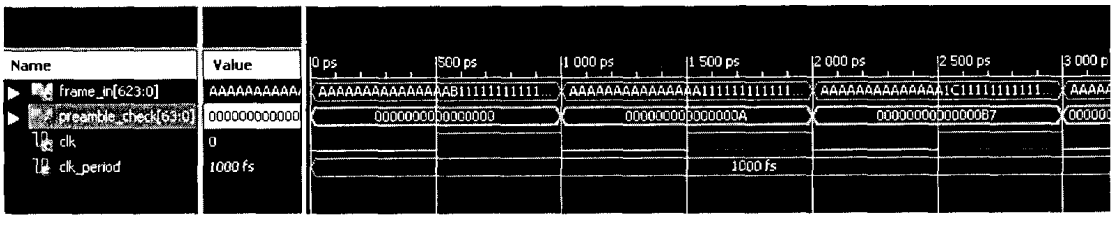

Figure 5.2: Preamble Check

The next block to test is the CRC Check block. Basically, this block checks the incoming frame's payload whether it matches the values stored in the CRC field of that specific frame. The CRC Check block uses multiple adders which are capable of wrapping around the last bit in case of overflow; hence the name Wrapper Adder. Figure 5.3 illustrates the functionally of this block. The results are obtained on port s; the first frame's CRC check is not correct and therefore the values on port s are not entirely ones. Similarly, the values on port s for the third, fourth, sixth, and seventh frames are not entirely ones which indicates mismatching between payloads' checksums and the CRC stored values. For the second and fifth frames, the CRC values are all ones indicating correct CRC Check.

Once a frame passes the preamble check and the CRC check, it enters the Frame Ripper block. As its name suggests, this block rips the frame into two pieces as shown

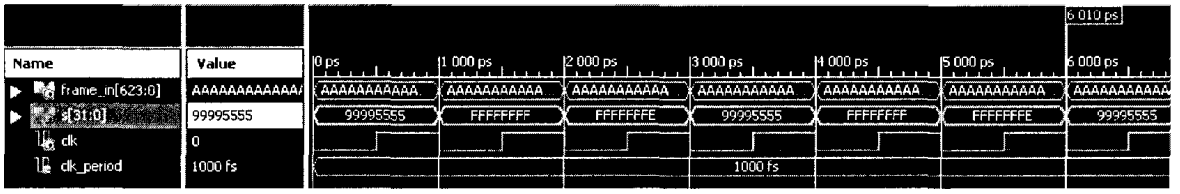

Figure 5.3: CRC Check

in Figure 4.4. In fact, this block sends the payload to be stored in the MEMORY block meanwhile the system performs scheduling processes. Those scheduling processes are carried out on the Frame Spec which contains both the frame's destination address and the source address as well as the type of this frame (voice, video, or best effort). In order to retrieve the payload once scheduling processes are done, the Frame Ripper generates a Payload Tag number for each payload. The Payload Tags are also included in the Frame Spec which is sent to the Scheduler. Payload Tags are also used in the MEMORY block as RAM addresses for the incoming payloads.

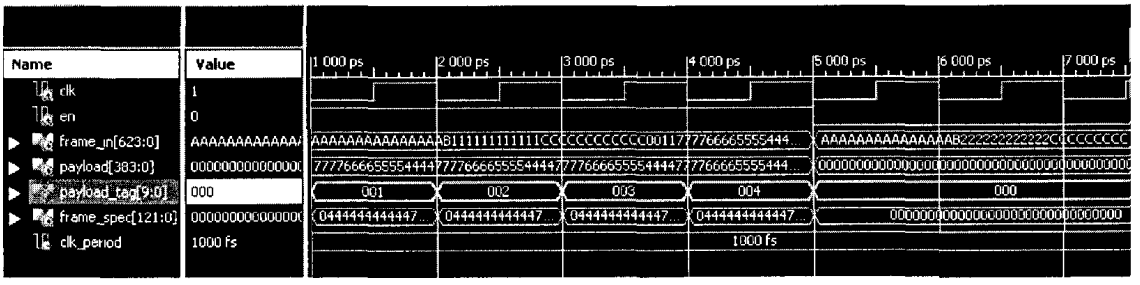

Figure 5.4: Frame Ripper

This block is tested to observe its response for incoming frames as shown in Figure 5.4. It's worth noticing that there is an *en* port which serves as enabler of this block; in other words, this port activates the block once the preamble and CRC checks are performed and the results are accepted. It can be seen that all output ports (*payload*, *payload_tag*, and *frame_spec*) are reset once the *en* port is zero; i.e., the block is deactivated. It can be also observed from Figure 5.4 that the *payload_tag* acts as a counter for the incoming payloads. That is, once a new payload (new frame) arrives, this block generates a new tag for this payload in order to store it in the MEMORY block.

5.2.1.2 MEMORY Block

Figure 5.5 illustrates how payloads are stored in the MEMORY block. The system is designed to pick the RAM address from *payload_out_tag* port when *we* port is set to zero, while it picks the address from *payload_in_tag* port when *we* port is set to one. In the first five clock cycles, the port *we* is set to zero; i.e., the MEMORY is reading whatever is stored in the associated RAM address. In the next five clock cycles, the

port *we* is set to one which indicates that the MEMORY is storing whatever comes from the port *payload_in* in the assigned address. Similar operations are performed on the next 20 clock cycles.

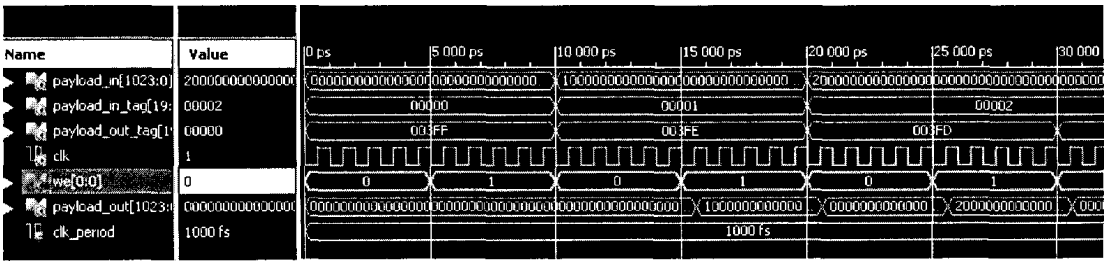

Figure 5.5: MEMORY Block Performing Writing Operation

The next figure shows how payloads are read on the *payload_out* port. In Figure 5.6, during first five clock cycles, the MEMORY block reads what was stored on the associated RAM address, which was in fact a payload of zeros. The next five cycles, it writes to another address a payload starts with two. Then, the next ten cycles reads what was sorted on the MEMORY's address provided by *payload_out_tag* port.

5.2.1.3 Scheduler

Figure 5.7 illustrates how the Classifier works. To test this block, four Frame Spec's are fed at a time. The block should put each Frame Spec on the correct port; i.e., voice Frame Spec should be put on *frame_vo_out* and so on. Since the block receives four Frame Spec's at a time, its output ports are also capable of carrying four Frame Spec's at a time. In case there was no Frame Spec to be filled, the block inserts zeros instead as shown on ports *frame_vo_out* and *frame_vi_out* in the first five cycles.

Frames sent from the Classifier enter the Prioritize on three ports along with their flag ports; *frame_vo_in* with *vo_flag* and so on. In order to test the Prioritize block, two different input signals were fed at this block's ports as follows:

1. First test patch; Figure 5.8

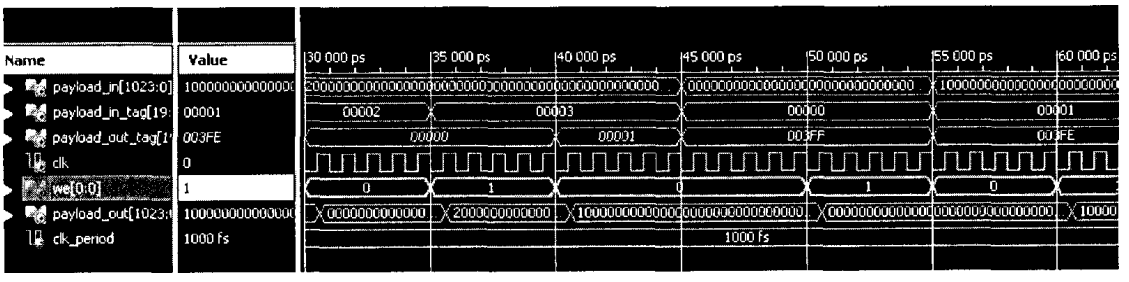

Figure 5.6: MEMORY Block Performing Reading Operation

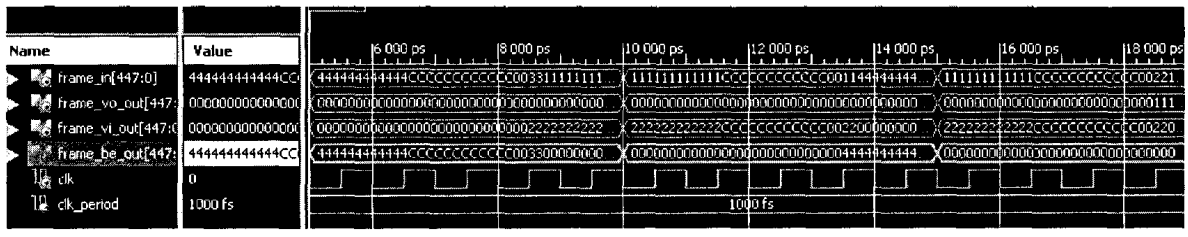

Figure 5.7: Classifier Output

- *frame_vo_in[447:336]* = zeros
- *frame_vo_in[335:224]* = x044444444444444444444444444444
- *frame_vo_in[223:0]* = zeros
- *vo_flag* = 0100
- *frame_vi_in[447:336]* = x011111111111111111111111111111
- *frame_vi_in[335:224]* = zeros
- *frame_vi_in[223:112]* = x033333333333333333333333333333
- *frame_vi_in[111:0]* = zeros
- *vi_flag* = 1010
- *frame_be_in[447:112]* = zeros
- *frame_be_in[111:0]* = x022222222222222222222222222222
- *be_flag* = 0001

2. Second test patch; Figure 5.9

- *frame_vo_in[447:224]* = zeros
- *frame_vo_in[223:112]* = x0BBBBBBBBBBBBBBBBBBBBBBBBBBB
- *frame_vo_in[111:0]* = zeros
- *vo_flag* = 0010
- *frame_vi_in[447:336]* = x0DDDDDDDDDDDDDDDDDDDDDDDDDDD
- *frame_vi_in[335:112]* = zeros
- *frame_vi_in[111:0]* = x0AAAAAAAAAAAAAAAAAAAAAAAAAAA
- *vi_flag* = 1001
- *frame_be_in[447:336]* = zeros
- *frame_be_in[335:224]* = x0CCCCCCCCCCCCCCCCCCCCCCCCCCC
- *frame_be_in[223:0]* = zeros
- *be_flag* = 0100

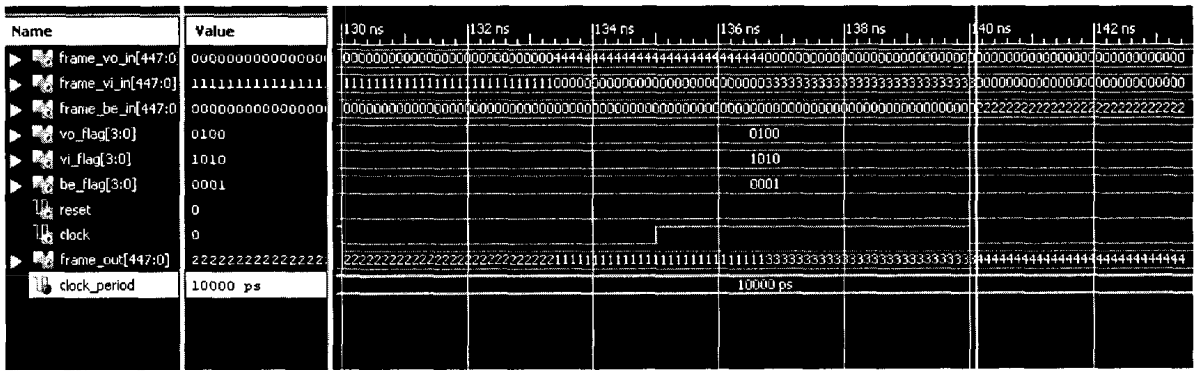

Figure 5.8: Prioritize Output - First Test

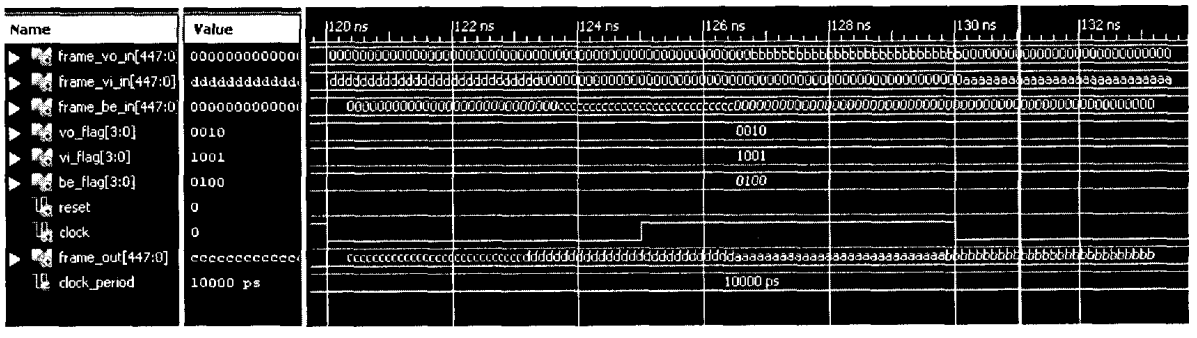

Figure 5.9: Prioritize Output - Second Test

5.2.1.4 Frame Constructor

The last subsystem to test is the Frame Constructor; however, as seen in Figure 4.7, the components used are mostly the same used for the Controller. Therefore, only the Frame Shaper is examined. Test performed on this block is shown in Figure 5.10; the *frame_out* port carries the outgoing frame after adding the preamble, CRC, destination, and source addresses.

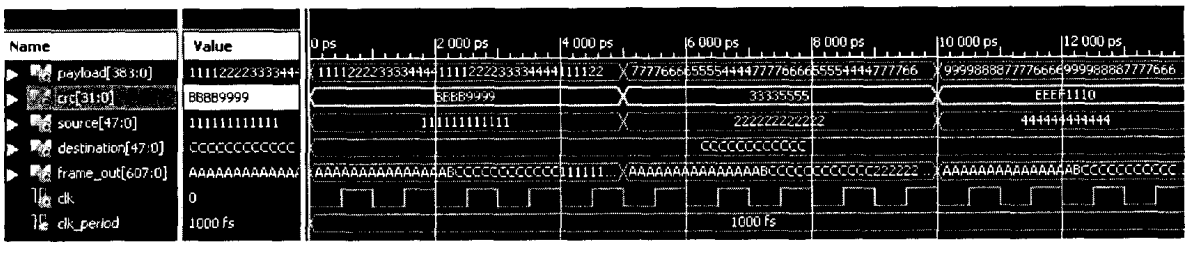

Figure 5.10: Frame Constructor Test

5.2.2 Results Discussion

Since the purpose of this design approach is to test the system through testing each element in it individually, components were fed their inputs and tested with only focus on having the accurate output. In light of that, all components in each subsystem were in fact performing their tasks correctly. However, it's worth noticing that the Prioritize had a remarkable initial response delay in both of the two tests were performed on it. In the first test, the Prioritize needed 95 ns to start responding, while it needed 115 ns in the second test.

This delay could actually be explained through the Prioritize design method itself. The Prioritize is designed using an FSM, in which 12 states were used. Each state was dedicated to check one segment of the four segments in each incoming frame (three incoming frames; voice, vidoe, and best effort). If the associated flag bit for a specific incoming segment is set high, then the next available segment on the output frame will be reserved for that incoming segment. This leads to checking every single incoming segment on the three incoming frame ports. For that, the system will need to go through 12 different states (in which each will require a clock cycle to be performed) leading to a remarkable initial delay.

5.3 The Conceptual Approach Results

5.3.1 Results Presentation

In the previous chapter, the conceptual approach system design proposes that the Ethernet Controller receives the incoming frames and then separates the payload from the other frame components in order to store the payload. This design consists of two main subsystems, along with the memory. To test the system, each subsystem is first tested individually to ensure that it works properly and then the integrated system is tested. At the end, the synthesis report is provided to examine the board's utilization of this design.

5.3.1.1 Testing the Ethernet Controller

The Ethernet Controller is tested by applying predefined values on specific ports. Those two ports represent the sent frame from the upper layer to the MAC layer, and the frame received from the physical layer to the MAC layer; *clientemactxd* and *phyemacrxd* respectively. Figure 5.11 is a snapshot of the resulted simulation. The purpose of this simulation is just to ensure that the Ethernet Controller is delivering frames correctly.

As shown in the Figure 5.11, a reset is applied for the first clock cycle to clear all stored values from previous simulations. The system starts to respond after 11.6 ns, i.e. when *emacphytxen* signal goes high. By that time, the physical layer starts

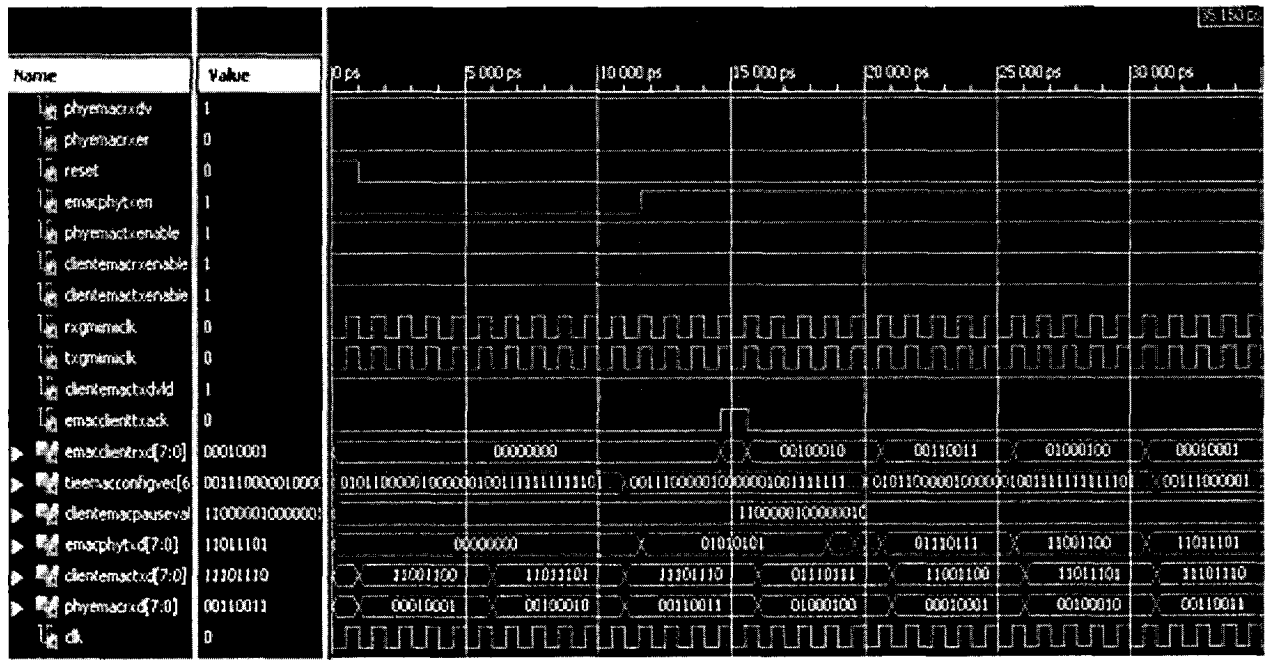

Figure 5.11: Ethernet Controller Performance

to receive frames sent from the upper layer through the MAC core. Those frames are received at *emacphytxd* port and were fed initially from *clientemactxd* port as follows:

- at 1 ns = 11001100.
- at 6 ns = 11011101.
- at 11 ns = 11101110.
- at 16 ns = 01110111.

After 15.6 ns, the frames received from the physical layer are seen at the upper layer through *emacclientrxd* port; these frames were fed initially from *phyemacrxd* port. The actual testing values sent on *phyemacrxd* port are as follows:

- at 1 ns = 00010001.
- at 6 ns = 00100010.
- at 11 ns = 00110011.
- at 16 ns = 01000100.

It is worth noticing that the system is unstable for the first 25.6 ns, i.e., the values fed to the system are observed correctly at the output ports after 25.6 ns.

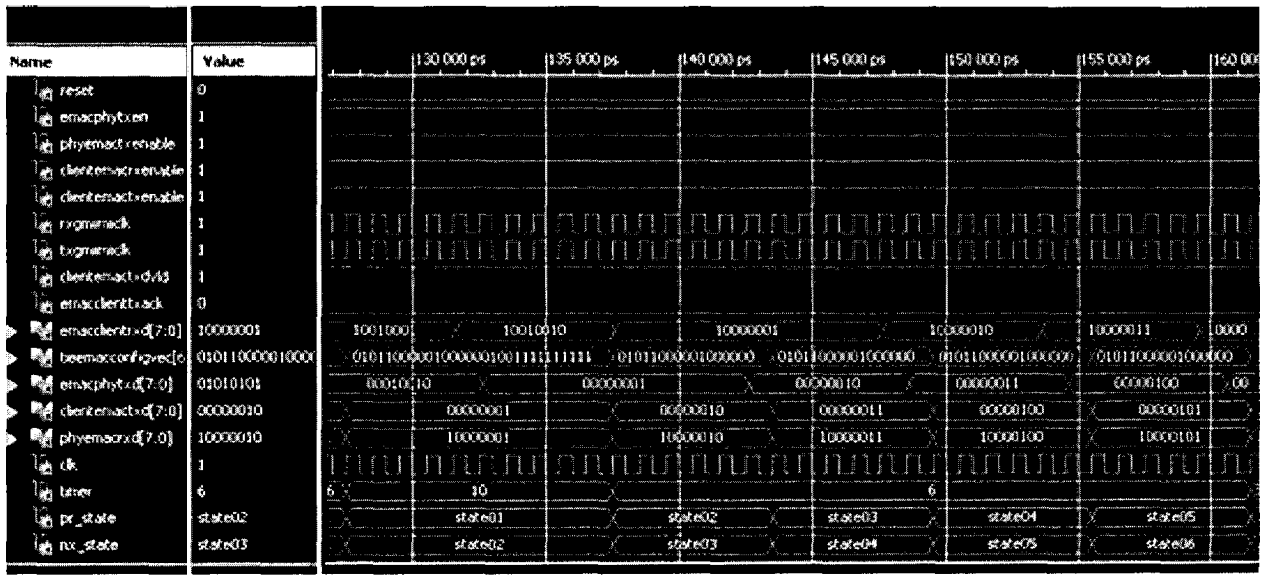

Figure 5.12: Beacon and Voice Downloading States

5.3.1.2 Testing the Scheduler

As described in previous chapter, the Scheduler is designed to have time constraint at each state. Therefore, to test the Scheduler, a reset state is forced at a specific time to observe the response of the system. Also, transition signals are assigned different values to distinguish the behavior at each state. For that, the following assumptions are made:

- The clock period (clk_period) is 1 ns.
- Transition signals are as follows:
  - time_beacon = 10,
  - time_VO = 6,
  - time_VIBE = 8,
  - time_BEu = 7,
  - time_VOu = 6,
  - time_max = 10.
- A reset is applied at t = 250 ns.

The key element to control the Ethernet Controller's behavior is the *tieemac-configvec* port; it actually contains specific bits in which RX or TX operations can be reset, half duplex mode be chosen, or MAC speed to be configured. As shown in Appendix A, bits 48 to 67 are responsible to set the Ethernet Controller configuration.

In this test, bits 48 to 67 of *tiemacconfigvec* are chosen as follow: (x058204). This choice allows the Ethernet Controller to provide full duplex connectivity, enabling both RX and TX, MAC speed at 1 Gbps.

Figure 5.12 is showing the beacon and voice downloading states, State01-State05. It can be observed that the timer has different value in beacon state than the one it has when it's in the voice downloading states. The port *clientemactxd* is assigned "0000 0001" while port *phyemacrxd* is assigned "1000 0001" during beacon state. As seen in the figure, the output signals are observed on *emacphytxd* and *emacclientrxd*, respectively. Similar to the beacon state, in the voice states the output signals are observed on *emacphytxd* and *emacclientrxd*, for the following input values:

- *clientemactxd* = 00000010 and *phyemacrxd* = 10000010
- *clientemactxd* = 00000011 and *phyemacrxd* = 10000011
- *clientemactxd* = 00000100 and *phyemacrxd* = 10000100
- *clientemactxd* = 00000101 and *phyemacrxd* = 10000101

In the next figure, Figure 5.13, the system moves to video and best effort downloading states just after State05 (as the FSM suggests). The input values fed to the systems are as follows:

- *clientemactxd* = 00000110 and *phyemacrxd* = 10000110
- *clientemactxd* = 00000111 and *phyemacrxd* = 10000111
- *clientemactxd* = 00001000 and *phyemacrxd* = 10001000
- *clientemactxd* = 00001001 and *phyemacrxd* = 10001001

The system continues to move to the next states according to the FSM model. Figure 5.14 shows the best effort uploading states (State10-State13). Also, the system's output values are observed on *emacphytxd* and *emacclientrxd* for the following input values:

- *clientemactxd* = 00001010 and *phyemacrxd* = 10001010
- *clientemactxd* = 00001011 and *phyemacrxd* = 10001011
- *clientemactxd* = 00001100 and *phyemacrxd* = 10001100
- *clientemactxd* = 00001101 and *phyemacrxd* = 10001101

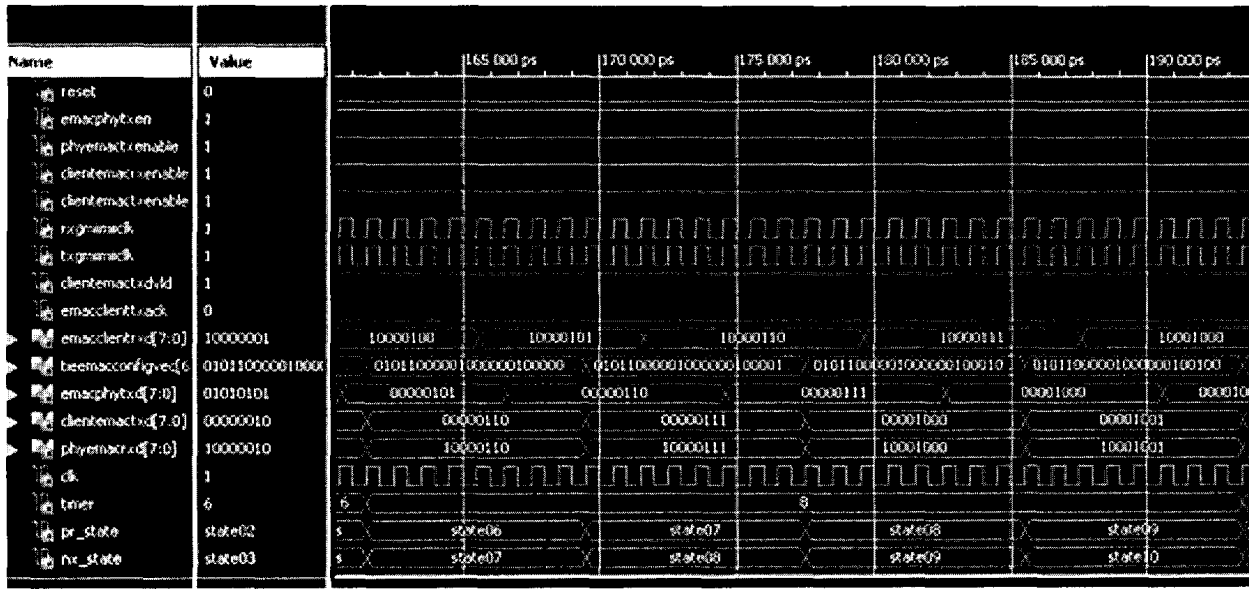

Figure 5.13: Video and Best Effort Downloading States

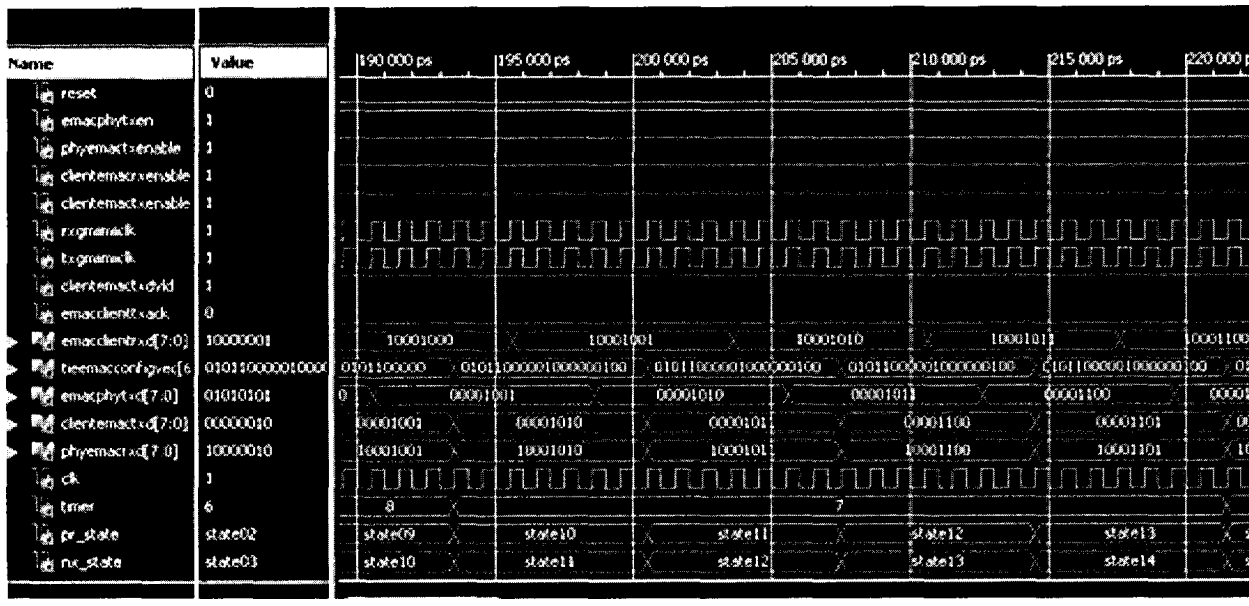

Figure 5.14: Best Effort Uploading States

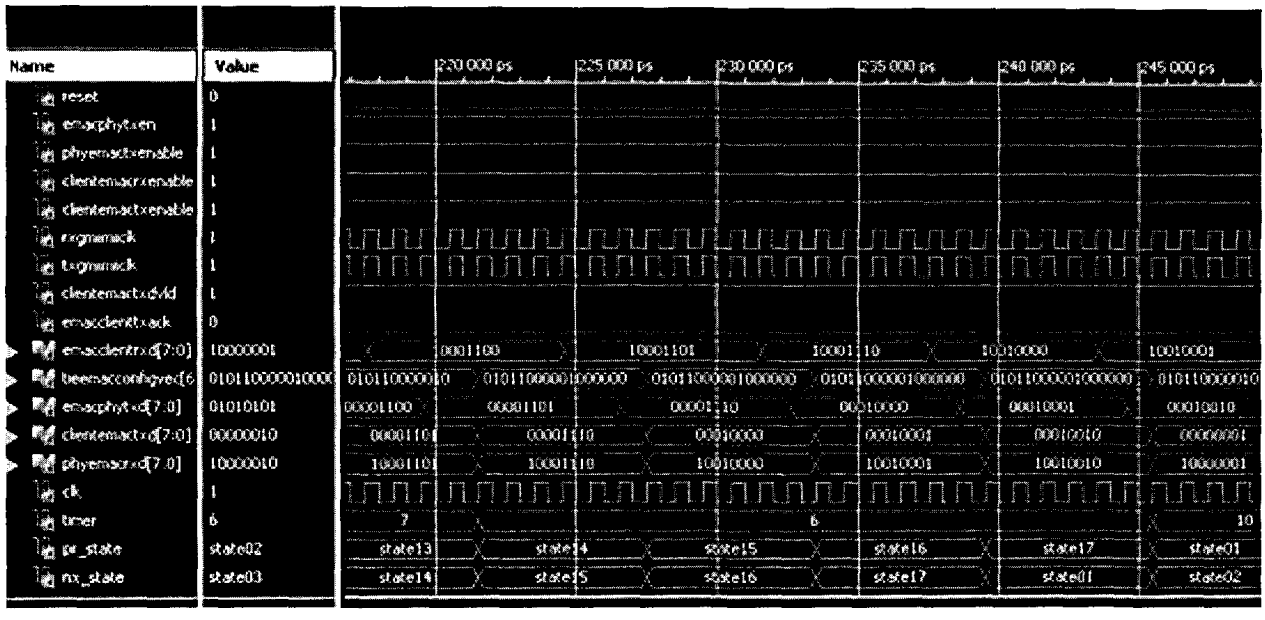

Figure 5.15: Voice Uploading States

After that, the system moves to voice uploading states (State14-State17) as shown in Figure 5.15. In these states, the system is fed the following inputs:

- *clientemactxd* = 00001110 and *phyemacrxd* = 10001110
- *clientemactxd* = 00010000 and *phyemacrxd* = 10010000
- *clientemactxd* = 00010001 and *phyemacrxd* = 10010001
- *clientemactxd* = 00010010 and *phyemacrxd* = 10010010

After all voice frames are uploaded, the system goes back to State01. To test the system's response to the reset state, a reset is forced as shown in Figure 5.16. In that figure, the system was at State01, but once a reset was applied, it moved to State00 (reset state) and it can be seen that controlling signals went low. Those signals are: *emacphytxen*, *phyemactxenable*, *clientemacrxenable*, *clientemactxenable*, *clientemactxdvld*, and *emacclienttxack*. Finally, the utilization report is presented below in Table 5.2, further details are shown in Appendix B.2.

5.3.2 Results Discussion

This design approach has been tested for various transmission states, both the downloading and uploading states. As seen in the previous section, the Ethernet Controller and the Scheduler performed as expected in terms of transmitting frames correctly. However, it can be observed that there is a delay in the Ethernet Controller's response

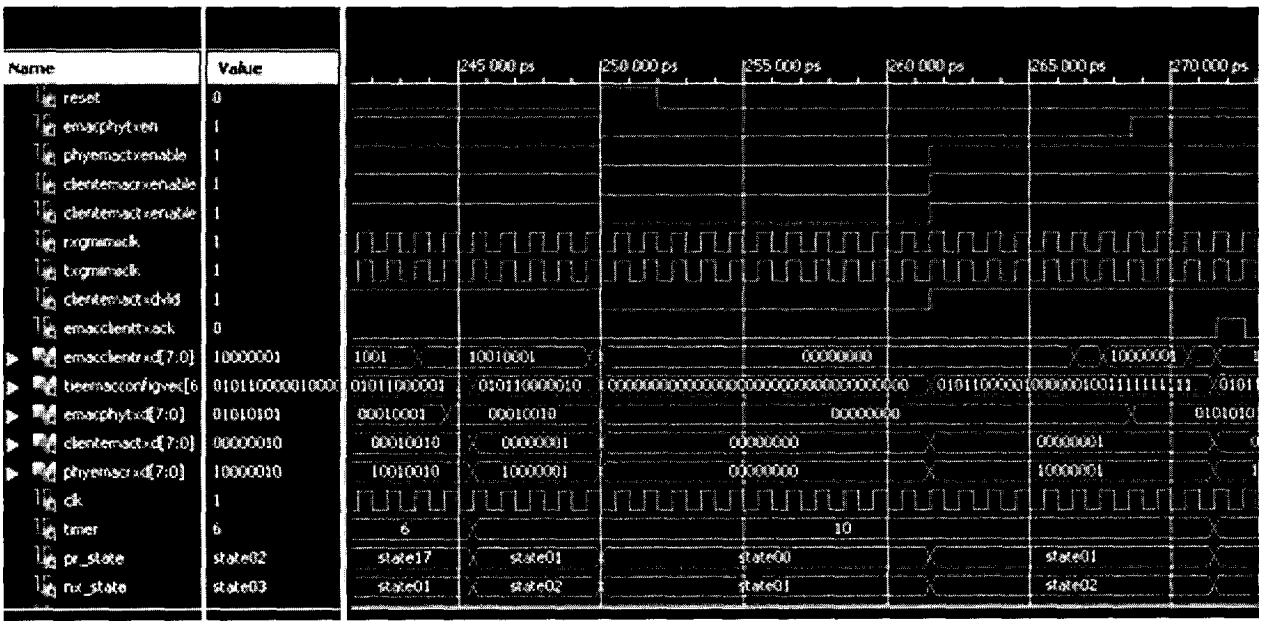

Figure 5.16: The System Moves to Reset State after reset is Asserted

Table 5.2: FPGA Usage Report - Device Utilization for the Conceptual Design

Slice Logic Utilization	Available	Used (Utilization)
Number of Slice Registers	69,120	967 (1%)
Number of Slice LUTs	69,120	859 (1%)
Number of occupied Slices	17,280	403 (2%)
Number of bonded IOBs	640	203 (31%)

especially as the system starts to receive input frames. This delay can be looked at through the core used to create this controller.

According to [40], the core is expected to have both transmit and receive path latencies. In the transmit path, the core tends to have maximum of 14 clock cycles latency; while in the receive path, the maximum latency can be 22 clock cycles. Moreover, there is a variation in latency of three clock cycles due to the crossing of clock domains within the core. These latency figures justifies the instability of the Ethernet Controller's performance at the initial stage of transmission.

Chapter 6
Future Work

This chapter turns the focus towards a future approach to schedule prioritized tasks using a kernel running on MicroBlaze soft processor. Xilinx embedded processors (MicroBlaze and PowerPC processors) can utilize a modular kernel highly integrated with XPS and received with EDK software package. This short chapter covers mainly Xilkernel through three sections in which the first section introduces Xilkernel key features and functionalities. The second section briefly discusses the methods to customize Xilkernel to suit targeted applications. The third section briefly discusses how Xilkernel could be utilized to run TP-TDMA protocol.

6.1 What is Xilkernel

As Xilinx defines Xilkernel, "it is a small, robust, and modular kernel"[41]. Xilkernel is highly integrated with the Platform Studio framework and is a free software library that is included in Xilinx Embedded Development Kit (EDK). Xilkernel allows designers a high degree of customization to achieve most optimum levels of size and functionality. Moreover, it's compatible with MicroBlaze, PowerPC 405, and PowerPC 440 processors. Xilkernel IPC services can be used to implement higher level services (such as networking, video, and audio) and subsequently run applications using these services.

Xilkernel includes the following key features:

- It improves the scalability level by allowing functionality inclusion or exclusion for targeted system.
- Quick complete kernel configuration and deployment within minutes from inside of Platform Studio.
- Static thread creation that startup with the kernel.
- System call interface to the kernel.
- Exception handling for the MicroBlaze processor.
- Memory protection using MicroBlaze Memory Management (Protection) Unit (MMU) features when available.

6.2 Customizing Xilkernel

Xilkernel is highly customizable, users can change the modules and individual parameters to suit their application. XPS Software Platform Settings dialog box provides an easy access to configuration settings for Xilkernel parameters. For a module to be customized in the kernel, a parameter with the name of the category set to TRUE must be defined in the Microprocessor Software Specification (MSS) file. A pthread can be customized as follows:

```
parameter config_pthread_support = true
```

If a configurable config_parameter for the module is not defined, that module will not be implemented. Commonly, these parameters and values are not manually set. In fact, once the information in the Software Platform Settings dialog box is entered, XPS generates the corresponding Microprocessor Software Specification (MSS) file entries automatically. The following is an MSS file snippet for configuring OS Xilkernel for a PowerPC processor system. The values in the snippet are sample values that target a hypothetical board [41]:

```
BEGIN OS
PARAMETER OS_NAME = xilkernel
PARAMETER OS_VER = 3.00.a
PARAMETER STDIN = RS232
PARAMETER STDOUT = RS232
PARAMETER proc_instance = ppc405_0
PARAMETER config_debug_support = true
PARAMETER verbose = true
PARAMETER systmr_spec = true
PARAMETER systmr_freq = 100000000
PARAMETER systmr_interval = 80
PARAMETER sysintc_spec = system_intc
PARAMETER config_sched = true
PARAMETER sched_type = SCHED_PRIO
PARAMETER n_prio = 6
PARAMETER max_readyq = 10
PARAMETER config_pthread_support = true
PARAMETER max_pthreads = 10
PARAMETER config_sema = true
PARAMETER max_sem = 4
PARAMETER max_sem_waitq = 10
PARAMETER config_msgq = true
PARAMETER num_msgqs = 1
PARAMETER msgq_capacity = 10
```

```
PARAMETER config_bufmalloc = true
PARAMETER config_pthread_mutex = true
PARAMETER config_time = true
PARAMETER max_tmrs = 10
PARAMETER enhanced_features = true
PARAMETER config_kill = true
PARAMETER mem_table = ((4,30),(8,20))
PARAMETER static_pthread_table = ((shell_main,1))
END
```

6.3 Utilizing Xilkernel for TP-TDMA Protocol

It was noticed in the proposed systematical approach that the Scheduler's design (mainly the Prioritize component) could have been modified to improve the initial delay. However, that can just be a starting point to radically change the design from using a hardware-based programming language (VHDL) to a software-based approach using MicroBlaze soft processor and its kernel, Xilkernel.

The limitations the design suffered from could have been avoided if the scheduler was designed using MicroBlaze and Xilkernel. Using Xilkernel, the scheduler could be designed in two different scheduling modes, either round-robin or priority-driven. In round-robin mode, frames can be scheduled in a time-slice form through a single ready queue in which each process is implemented during a configured time slice before executing the next process in the queue. In priority-driven mode, the process that is at the top of the ready queue with the highest priority is executed first and so on. The highest priority is always assigned 0.

In the hypothetical example in Section 6.2, the system is configured to use a priority-driven mode with 6 different levels of priorities.The maximum number of processes that can be active in the ready queue is 10. The maximum number of software timers that the processor can use in this kernel is 10. These settings can only be activated if the associated parameter is set true, as follows:

```
PARAMETER config_sched = true
PARAMETER sched_type = SCHED_PRIO
PARAMETER n_prio = 6
PARAMETER max_readyq = 10
PARAMETER config_time = true
PARAMETER max_tmrs = 10
```

Chapter 7
Conclusion

Triple-Play services are attracting more researchers to engage in investigating the possibilities of integrating these services and, as a result, reducing operation costs. To reach smooth integration, FPGA boards seem to be the most suitable platform as they have high levels of flexibility and customization. Therefore, choosing Xilinx Virtex-5 board offers a comfortable platform to implement the TP-TDMA protocol; in addition, it allows the testing of a new approach through using a soft processor on the board.

In summary, the work of this thesis took advantage of reconfigurable platforms to implement the TP-TDMA protocol; however, reconfigurable platforms can be used to implement a wide range of other protocols. The contributions achieved in this thesis can be listed as follows:

- Creating a MicroBlaze soft processor to test the board in which the processor was then deployed on a Virtex-5 board. That deployment was achieved through defining the accurate pin assignment on the FPGA. The main objective of creating a MicroBlaze soft processor was to test the board's connectivity with the computer and to observe the communication results on the Hyper Terminal. It was observed that MicorBlaze's consumption is relatively low, as only 2% of the board's Registers and LUTs was consumed.

- Designing two different approaches to implement TP-TDMA protocol; the systematic and the conceptual approaches. In the systematic approach, the tests were made on each component showed that the proposed design performs the required scheduling tasks as suggested by TP-TDMA protocol. Similarly, in the conceptual approach, the overall tests performed on the scheduler showed that this design also meets the proposed TP-TDMA protocol requirements.

- On the two proposed design approaches, initial response delays were observed. In the systematic approach, the design required the Prioritize to check every single incoming segment of the three incoming frame ports leading to forcing the system to go through 12 different states. While in the conceptual approach, the core used to create the Ethernet Controller has 14 clock cycles and a 22 clock cycles latency in the transmit and receive paths, respectively.

References

[1] Anticipating opportunity in the triple-play market: Issues, possibilities and importance of standards-based technology. [Online]. Available: www.visionael.com/solutions/whitepapers/whitepaper_tripleplay_06.pdf

[2] D. Srefjord, S. Lindroos, and L. Karlsson. Triple play - a strategy for the convergence of home electronics in the swedish market. [Online]. Available: http://www.cse.chalmers.se/tsigas/Courses/DCDSeminar/Files/triple_play_strategydatakomm_report.pdf

[3] C. Hellberg, G. Dylan, and T. Boyes, *Broadband Network Architectures: Designing and Deploying Triple-Play Services*, 1st ed. Prentice Hall, 2007.

[4] The perfect storm: Why video conferencing will dominate business communications. [Online]. Available: http://www.glgroup.com/News/13329.html

[5] An emerging triple play: Video communications, managed services and unified communications. [Online]. Available: www.pgi.com/us/en/Brockmann_Tripleplay_Report.pdf

[6] P. Kampanakis, M. Kallitsis, S. Sridharan, and M. Devetsikiotis. (2006) Triple Play - A Survey. [Online]. Available: http://www4.ncsu.edu/~mgkallit/files/3PReportTechnical1.4.pdf

[7] Developing a holistic testing strategy for ensuring a successful iptv deployment. [Online]. Available: http://www.ixiacom.com/pdfs/library/white_papers/iptv_dev_holistic_testting_strategy.pdf

[8] Advantages of field programmable gate arrays. [Online]. Available: www.men.de/docs-ext/expertise/pdf/fpga_advantages.pdf

[9] J. Kurose and K. Ross, *Computer networking: a top-down approach*, 4th ed. Boston, MA: Pearson/Addison-Wesley, 2008.

[10] J. Barcelo, "CSMA/ECA: Carrier Sense Multiple Access with Enhanced Collision Avoidance," Ph.D. dissertation, Universitat Pompeu Fabra, Jan. 2009.

[11] J. Choi, J. Yoo, S. Choi, and C. Kim, "Eba: an enhancement of the ieee 802.11 dcf via distributed reservation," *Mobile Computing, IEEE Transactions on*, vol. 4, no. 4, pp. 378 – 390, july-aug. 2005.

[12] D. Niyato and E. Hossain, "Queue-aware uplink bandwidth allocation and rate control for polling service in ieee 802.16 broadband wireless networks," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 6, pp. 668 – 679, june 2006.

[13] K. Wongthavarawat and A. Ganz, "Packet scheduling for QoS support in IEEE 802.16 broadband wireless access systems," *International Journal of Communication Systems*, vol. 16, no. 1, pp. 81–96, 2003.

[14] H. Alavi, M. Mojdeh, and N. Yazdani, "A quality of service architecture for ieee 802.16 standards," in *Communications, 2005 Asia-Pacific Conference on*, oct. 2005, pp. 249 253.

[15] X. Bai, A. Shami, and Y. Ye, "Robust qos control for single carrier pmp mode ieee 802.16 systems," *Mobile Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 416 429, april 2008.

[16] D. Dechene and A. Shami, "Experimental triple-play service delivery using commodity wireless lan hardware," in *Communications, 2009. ICC '09. IEEE International Conference on*, june 2009, pp. 1 –5.

[17] *Embedded Systems Development: Participant Guide*, Xilinx, 2010, embd21000-12-wkbp-rev1.

[18] *ML505/ML506/ML507 Evaluation Platform - User Guide*, Xilinx, Oct. 2009, ug347.

[19] M. Zaidi, J. Bhar, R. Ouni, and R. Tourki, "A new solution for micro-mobility management in 802.11 wireless lans using fpga," in *Signals, Circuits and Systems, 2008. SCS 2008. 2nd International Conference on*, nov. 2008, pp. 1 –7.

[20] S. Georgi, C. Prieske, and H. Rohling, "Cross layer hardware demonstrator for wireless communication based on a fpga," in *Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*, sept. 2009, pp. 9 13.

[21] H. Yu, K. Song, K. Ryu, Y. Kim, S. Min, and S. kyu Lee, "Design and fpga implementation of mimo-ofdm based wlan systems," in *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, vol. 3, may 2006, pp. 1333 1338.

[22] Y. Kim, H. Jung, H. H. Lee, and K. R. Cho, "Mac implementation for ieee 802.11 wireless lan," in *ATM (ICATM 2001) and High Speed Intelligent Internet Symposium, 2001. Joint 4th IEEE International Conference on*, 2001, pp. 191 195.

[23] T. Shono, H. Shiba, Y. Shirato, K. Uehara, K. Araki, and M. Umehira, "Performance of ieee 802.11 wireless lan implemented on software defined radio with hybrid programmable architecture," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 3, may 2003, pp. 2035 - 2040.

[24] A. Bhavikatti and S. Kulkarni, "Vhdl modeling of wi-fi mac layer for transmitter," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, march 2009, pp. 1 -5.

[25] T. Y. Tse, "Wimedia uwb mac layer implementation in fpga," in *Wireless and Mobile Communications, 2009. ICWMC '09. Fifth International Conference on*, aug. 2009, pp. 234 -238.

[26] S. Nabi, C. Wells, and W. Vanderbauwhede, "A dynamically reconfigurable system-on-chip for implementing wireless macs," in *Research in Microelectronics and Electronics Conference, 2007. PRIME 2007. Ph.D.*, july 2007, pp. 37 40.

[27] S. Ji, W. H. Kim, C. Chung, and J. Kim, "A zigbee compliant baseband and mac processor," in *SOC Conference, 2007 IEEE International*, sept. 2007, pp. 87 -90.

[28] J. Hui and M. Devetsikiotis, "Designing improved mac packet schedulers for 802.11e wlan," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 1, dec. 2003, pp. 184 - 189.

[29] J. Lazanyi, "Instruction set extension using microblaze processor," in *Field Programmable Logic and Applications, 2005. International Conference on*, aug. 2005, pp. 729 730.

[30] M. Vetromille, L. Ost, C. Marcon, C. Reif, and F. Hessel, "Rtos scheduler implementation in hardware and software for real time applications," in *Rapid System Prototyping, 2006. Seventeenth IEEE International Workshop on*, june 2006, pp. 163 168.

[31] J. Kadlec, M. Danek, and L. Kohout, "Proposed architecture of configurable, adaptable soc," in *Signals and Systems Conference, 208. (ISSC 2008). IET Irish*, june 2008, pp. 368 -373.

[32] E. Wang, S. Zhang, Q. Hu, J. Yi, and X. Sun, "Implementation of an embedded gps receiver based on fpga and microblaze," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, oct. 2008, pp. 1 -4.

[33] G. Stewart, D. Renshaw, and M. Riley, "A low-cost, fpga based, video streaming server," in *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, feb. 2007, pp. 187 -190.

[34] *Embedded Systems Development Lab Workbook: MicroBlaze Processor and Spartan-6 FPGA SP605 Board*, Xilinx, 2010, embd21000-12-wkb-lab-sp605-rev1.

[35] *Tri-Mode Ethernet MAC v4.1*, Xilinx, Apr. 2009, ds297.

[36] *Tri-Mode Ethernet MAC v4.3 - User Guide*, Xilinx, Dec. 2009, ug138.

[37] N. Zainalabedin, *VHDL: Modular Design and Synthesis of Cores and Systems*, 3rd ed. McGraw Hill, 2007.

[38] P. Ashenden, *The Student Guide to VHDL*, 2nd ed. Morgan Kaufman, 2008.

[39] V. Pedroni, *Circuit Design with VHDL*, 1st ed. MIT Press, 2004.

[40] *Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC - User Guide*, Xilinx, Feb. 2010, ug074.

[41] *Xilkernel (v5.00a)*, Xilinx, Jun. 2010, ug708.

Appendix A
Configuration Vector Details

Configuration vector is composed of 68 bits. Following table provides description of the purpose of each bit in this vector.

Table A.1: Configuration Vector Bits Description[36]

Bit	Description
47:0	Pause frame MAC Source Address [47:0]: This "MAC Address" is used by the MAC core to match against the destination address of any incoming flow control frames, and as the source address for any outbound flow control frames. The bits in this vector field are ordered so that the least significant bit of the MAC Address (IEEE802.3 definition) is stored in the least significant bit of this vector field. Consequently, bit 0 of this field will differentiate between an individual or group (multicast) address. The transmission order within a MAC frame is to send the least significant bit of the MAC Address first. Consequently, bits 7-0 of this vector field will represent the first byte to appear in frame transmission.
48	Receiver Half Duplex: If '1,' the receiver operates in half-duplex mode. If '0,' the receiver operates in full-duplex mode. If the TEMAC has been generated without half-duplex support then this input to the core will be unused.
49	Receiver VLAN Enable: When this bit is set to '1,' VLAN tagged frames are accepted by the receiver.
50	Receiver Enable: If set to '1,' the receiver block is operational. If set to '0,' the block ignores activity on the physical interface RX port.
51	Receiver In-band FCS Enable: When this bit is '1,' the MAC receiver will pass the FCS field up to the client as described in "Client-Supplied FCS Passing," on page 60. When it is '0,' the MAC receiver will not pass the FCS field. In both cases, the FCS field will be verified on the frame.

Bit	Description
52	Receiver Jumbo Frame Enable: When this bit is '0,' the receiver will not pass frames longer than the maximum legal frame size specified in IEEE 802.3-2005 ("Maximum Permitted Frame Length," on page 70). When it is '1,' the receiver will not have an upper limit on frame size.
53	Receiver Reset: When this bit is '1,' the receiver is held in reset. This signal is an input to the reset circuit for the receiver block.
54	Transmitter Interframe Gap Adjust Enable: If '1,' and the MAC is set to operate in full-duplex mode, then the transmitter will read the value of the clientemactxifgdelay port and set the Interframe Gap accordingly. If '0,' the transmitter will always insert at least the legal minimum interframe gap.
55	Transmitter Half Duplex: If '1,' the transmitter operates in half-duplex mode. If '0,' the transmitter operates in fullduplex mode. If the TEMAC solution has been generated without half-duplex support, this input to the core will be unused.
56	Transmitter VLAN Enable: When this bit is set to '1,' the transmitter allows the transmission of VLAN tagged frames.
57	Transmitter Enable: When this bit is '1,' the transmitter will be operational. When it is '0,' the transmitter is disabled.
58	Transmitter In-Band FCS Enable: When this bit is '1,' the MAC transmitter will expect the FCS field to be passed in by the client as described in "Client-Supplied FCS Passing," on page 67. When it is '0,' the MAC transmitter will append padding as required, compute the FCS and append it to the frame.
59	Transmitter Jumbo Frame Enable: When this bit is '1,' the MAC transmitter will allow frames larger than the maximum legal frame length specified in IEEE 802.3-2005 to be sent. When set to '0,' the MAC transmitter will only allow frames up to the legal maximum to be sent.
60	Transmitter Reset: When this bit is '1,' the MAC transmitter is held in reset. This signal is an input to the reset circuit for the transmitter block.
61	Transmit Flow Control Enable: When this bit is '1,' asserting the clientemacpausereq signal causes the MAC core to send a flow control frame out from the transmitter as described in "Transmitting a Pause Control Frame," on page 77. When this bit is '0,' asserting the clientemacpausereq signal will have no effect.

Bit	Description
62	Receive Flow Control Enable: When this bit is '1,' received flow control frames will inhibit the transmitter operation as described in "Receiving a Pause Control Frame," on page 78. When it is '0,' received flow frames are passed up to the client.
63	Length/Type Error Check Disable: When this bit is '1,' the core will not perform the length/type field error checks as described in "Length/Type Field Error Checks," on page 61. When it is set to '0,' the length/type field checks will be performed; this is normal operation.
64	Address Filter Enable: When this bit is '0,' the address filter is enabled. If it is set to '1,' the address filter will operate in promiscuous mode. If the TEMAC solution has been generated without the optional Address Filter, this input to the core will be unused.
66:65	MAC Speed Configuration: "00" - 10 Mbps "01" - 100 Mbps "10" - 1 Gbps When the TEMAC solution is generated for only 1 Gbps speed support, these inputs will be unused. When the TEMAC solution is generated for only 10 Mbps or 100 Mbps speed support, only bit 65 will be used to differentiate the speed: bit 66 will be unused.
67	Control Frame Length Check Disable: When this bit is set to '1,' the core will not mark control frames as 'bad' if they are greater than the minimum frame length.

Appendix B
Detailed Utilization Reports

B.1 Utilization Reports for MicroBlaze Design

Table B.1: FPGA Usage Report - Device Utilization for MicroBlaze Testing: Further Details

Slice Logic Utilization	Available	Used (Utilization)
Number of Slice Registers - Number used as Flip Flops - Number used as Latch-thrus	69,120	1,631 (2%) 1,627/1,631 4/1,631
Number of Slice LUTs - Number used as logic - Number used as Memory	69,120	1,879 (2%) 1,737/1,879 138/1,879
Number of occupied Slices	17,280	965 (5%)
Number of bonded IOBs - Number of LOCed IOBs	640	12 (1%) 12/12
Number of BlockRAM/FIFO - Number using BlockRAM only	148	8 (5%) 8/8
Number of BUFG/BUFGCTRLs	32	2 (6%)
Number of BSCANs	4	1 (25%)
Number of DSP48Es	64	3 (4%)
Number of PLL_ADVs	6	1 16%)

B.2 Utilization Reports for the Conceptual System Design

Table B.2: FPGA Usage Report - Device Utilization for the Conceptual Design: Further Details

Slice Logic Utilization	Available	Used (Utilization)
Number of Slice Registers - Number used as Flip Flops	69,120	967 (1%) 967/967
Number of Slice LUTs - Number used as logic - Number used as Memory	69,120	859 (1%) 845/859 13/859
Number of occupied Slices	17,280	403 (2%)
Number of bonded IOBs	640	203 (31%)

Appendix C
User Constraints File for MicroBlaze Design

```
Net fpga_0_RS232_RX_pin LOC=AG15;
Net fpga_0_RS232_TX_pin LOC=AG20;
Net fpga_0_LEDS_GPIO_IO_O_pin<0> LOC=H18;
Net fpga_0_LEDS_GPIO_IO_O_pin<1> LOC=L18;
Net fpga_0_LEDS_GPIO_IO_O_pin<2> LOC=G15;
Net fpga_0_LEDS_GPIO_IO_O_pin<3> LOC=AD26;
Net fpga_0_LEDS_GPIO_IO_O_pin<4> LOC=G16;
Net fpga_0_LEDS_GPIO_IO_O_pin<5> LOC=AD25;
Net fpga_0_LEDS_GPIO_IO_O_pin<6> LOC=AD24;
Net fpga_0_LEDS_GPIO_IO_O_pin<7> LOC=AE24;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC=AH15;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=E9;
```