Configurable Architecture for System-Level Prototyping of High-Speed Embedded Wireless Communication Systems

Visvanathan Subramanian

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Dr. Joseph G. Tront, Chair
Dr. Charles W. Bostian
Dr. Scott F. Midkiff

January 13, 2003
Blacksburg, Virginia

Keywords: Configurable Architecture, FPGA, LMDS, Rapid Prototyping, Communication System Design

Configurable Architecture for System-Level Prototyping of High-Speed Embedded Wireless Communication Systems

Visvanathan Subramanian

(ABSTRACT)

Broadband wireless technologies have the potential to provide integrated data and multimedia services in several niche areas. There is a growing need to develop high-performance communication systems that can satisfy high-end data processing requirements inherent in these technologies. The speed and complexity of these systems necessitates designers to break away from traditional architectures and design methodologies. A more comprehensive and demanding design and verification process including both hardware and software is required. Field-programmable gate arrays (FPGA) offer an attractive alternative to the low efficiency of Digital Signal Processor (DSP) based systems and low flexibility of Application Specific Integrated Circuits (ASIC). The availability of high-density, high-performance field-programmable gate arrays with several capabilities, like embedded memory and advanced routing, together with the adaptability that they offer make them highly desirable for developing hardware prototypes of communication systems.

This thesis describes the development of a configurable architecture and FPGA-based design methodology used in the development of a Local Multipoint Distribution Service (LMDS) gateway for a disaster response network. The design of the gateway posed several challenges due to high data rates (120 Mbits/sec) and adaptive features like variable Forward Error Correction Coding and optional link-level retransmissions. The design decisions and simulation results of the verification process are discussed in detail. Finally, the aspects of testing and integration of the prototype in the overall system are discussed.

Dedication

To my parents and all my teachers

Acknowledgements

A number of individuals deserve recognition for their role in helping me complete this thesis. I would like to acknowledge and offer my gratitude to all of them for their constant support and encouragement throughout this process.

To my parents, S.V. Subramanian and Gowri Subramanian, who persuasively encouraged me to follow my dreams and aspirations and who bolstered my confidence, during the various highs and lows I encountered. For their prayers and blessings, I am eternally grateful.

To my Advisor, Dr. Joseph Tront, who afforded me vast amounts of time and latitude while completing my studies. I am grateful for his direction and sage advice without which this work would not have been possible.

To Dr. Charles Bostian and Dr. Scott Midkiff, for their guidance, support and faith in me to complete this work.

To the Center for Wireless Telecommunications (CWT), Virginia Tech for the assistantship and support, that enabled me to complete my research and study at Virginia Tech. To the CWT faculty, staff and graduate researchers, for their valuable inputs and suggestions. I would like to gratefully acknowledge the grant from the National Science Foundation (Award #9983463) that made this research possible.

To Lockheed Martin Global Telecommunications (LMGT), and ADI Engineering for the technical input and support, during various phases of the design.

To my friends and family, for their constant love and support.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Overview

The wireless market place has tremendous growth potential and huge demand for solutions as users are discovering that wireless appliances contain increasing functionality that makes their jobs and their lives easier. These wireless devices are becoming more like true computing platforms that run applications, including Internet access, e-mail, multimedia messaging, synchronizing calendars over wireless networks, gaming and downloading music. This call for break-through products requires wireless system designers to remain at the forefront of technology and convert these expectations into reality.

The expanding wireless application space is raising device performance requirements, as large streams of voice, data, audio and video need to be processed on wireless communication devices. To boost revenues, service providers are also rushing to offer more data/application services on wireless devices, e.g., video services are now offered on cell phones. To meet the demand for ubiquitous Internet access and the ability to access and share information from anywhere, at any time, wireless appliances and the supporting networking infrastructure must be equipped with adequate computing and signal processing capabilities. For example, third-generation (3G) cell-phone handsets are estimated to require several thousands of MIPS of signal-processing horsepower, just to capture a signal and extract data packets from it. Increasingly, wireless terminals require signal-processing services not just to receive the packets but to act on the payload as well. For instance, broadband wireless data networking generally includes forward error correction (FEC) and encryption, which requires high-speed data manipulation at both ends of the wireless channel. The wide range of signal processing requirements - from simple cyclic redundancy checks (CRC) to complex CODECs – can be realized using a wide range of implementations from just software to optimized system-on-chip solutions.

The signal processing demands are only bound to increase into the future. To fully realize the wireless market's growth potential, wireless system designers need to employ emerging technologies that can enable innovative solutions while addressing the concomitant issues and risks.

1.2 Communication Systems Design Challenges

Wireless transmission is inherently limited by the available spectrum and impaired by path loss, interference, multi-path propagation, which all leads to potential problems like delay spread and fading. Consequently, designers of broadband wireless communication systems face several intricate issues related to access mechanisms, error rates, transmission rates and bandwidth. These challenges are further compounded in the case of fixed broadband wireless designs where the influence of wind, vehicular traffic, and foliage make for a hostile fading environment [1]. Therefore, the first major challenge is to design a wireless link in a fading environment to look like a wire line link so as to provide the same or similar quality of service as other competing broadband wired technologies.

An end-to-end approach to error control used in wired networks that ensures reliability by mechanisms in the end systems may not be suitable for their wireless counterparts. The unsuitability of the end-to-end approach stems from the fact that the unreliability of the media in wireless communications is the major cause for dropped packets, whereas congestion accounts for most of the packet losses in the wired domain. Instead, error recovery mechanisms such as FEC and automatic repeat request (ARQ) are used to guarantee reliability in the traversed wireless links. This solution adds additional complexity and computational workload to the designs.

The second major challenge is at the Medium Access Control (MAC) layer, where it is crucial that future MACs support sophisticated physical layer techniques such as adaptive modulation and coding or spatial multiplexing. The adaptive techniques thrust significant processing workload on the system implementations.

In general, wireless designs offer greater challenges than wired systems. Next-generation broadband wireless communication applications incorporate several features such as high-speed, large-bandwidth network and radio interfaces, complex digital blocks that implement multi-layer protocols, and significant amounts of embedded memory. High-level protocol descriptions have to be rapidly translated into hardware and software that realize the system. Rapid advances in process technology give us the ability, at least in theory, to design ever more complex communication systems capable of operating at higher speeds. But the design complexities, in conjunction with more involved device models that these processes require, create a design crisis where the development cycles and iteration times consume more and more effort and time. The designs push the limits of current EDA tools and a radical design flow throughput is needed to verify the design early in the design process. A thorough test and verification process to achieve timing closure and signal integrity must be completed within shrinking time-to-market windows. Moreover, the system design must be capable of adapting to late changes in specification or emerging standards so as to reduce the risk of costly hardware and software redesigns. A design strategy to meet these needs will be described in this thesis.

1.3 Research Goals

The primary goal of this work is to identify and explore configurable architectures that aid in rapid system-level prototyping of embedded wireless communication systems and is also suitable in the context of modern system level design methodologies. The intention here is to tradeoff some measure of density and performance to achieve reasonable design times and rapid system level prototyping. Implementation options with a high degree of adaptability that allows alterations even late into the design process are studied. Configurable devices like FPGAs are the basis for flexibility. Tolerance to modifications makes FPGAs highly desirable for developing hardware prototypes or marketable products for communication systems. Another implementation option that is explored is the trend of moving away from using general-purpose processors [2] in favor of custom processors or configurable system-on-chips (SOC). The custom solutions are

usually optimized towards a particular domain or constraint, for example, network processors for network router applications or low power processors for handheld devices.

Besides exploring the design space, an attempt is made to identify elements of the architecture space that are suitable for wireless communication design. In particular, a hybrid memory-centric re-configurable architecture is described combining traditional field-programmable gate array (FPGA) for low-level network protocols with domain specific processors for higher-level packet processing. The architecture offers greater design flexibility by simplifying interfaces and allowing the integration of heterogeneous hardware blocks.

1.4 Application

This work focuses on the architecture and design of the Gateway Controller for a high-speed Local Multi-point Distribution Service (LMDS) broadband wireless communication system to aid in emergency response and management. The system is described in detail in Chapter 2.

1.5 Thesis Organization

Chapter 2 introduces the concept of disaster response communications. It also provides an overview of the rapidly deployable disaster response communication system developed by Virginia Tech's Center for Wireless Telecommunications, in partnership with Science Applications International Corporation (SAIC). It then examines the components of the system and discusses some of its interesting features.

Chapter 3 addresses the issues for developing the system architecture by examining the existing and emerging system design methodologies. It also outlines the architectural design space for these systems and describes the design decisions and tradeoffs that were encountered.

Chapter 4 explains the hardware and software implementation of the design. The components of the hardware and software platforms are presented.

Chapter 5 explains the verification, integration and validation of the design. Very High Speed Integrated Circuit Hardware Description Language (VHDL) simulation results used to verify the hardware implementation early in the design process are presented.

Chapter 6 summarizes and concludes the thesis with recommendations for future research in this area.

CHAPTER 2

Informal Specification of System Requirements

2.1 Overview

Before embarking on an embedded wireless system design, we begin by examining the system requirements for a Local Multi-point Distribution Service (LMDS) broadband wireless communication system for disaster response communications. This chapter introduces the LMDS disaster response Gateway Controller and the application area of this research work, viz., disaster response communications. The intention is to acquaint the reader with the disaster response system developed at the Center for Wireless Telecommunications (CWT) at Virginia Tech for which the Gateway Controller is being designed. This is necessary to identify a set of services and applications that are to be supported and also gives an idea of the complexity and problem areas that must be addressed during the design.

2.2 Disaster Response Communications

Existing disaster response communications support primarily focuses on voice. However, data connectivity is rapidly becoming crucial because of the dependence on information technology (IT) based infrastructure integrated into modern disaster response systems. Broadband wireless communication technologies have the potential to provide the bandwidth necessary to support voice, data and video applications and content that are being developed for disaster response. Besides providing high-speed connectivity, the system must also be suitable for rapid deployment and remain robust even in adverse environmental conditions.

First responders to disasters, both man-made and natural, must be able to gather critical data and disseminate it using robust means. It is imperative that decision makers be able to request and receive this critical data, so that they may appropriately shape the

nature and scale of the disaster response. The field responders would also benefit from the knowledge base of archived information about the disaster area available on public or agency networks. For example, firefighters responding to a disaster can use Geographical Information Systems (GIS) based applications to find the location of fire hydrants that may be concealed in the rubble. The applications and possibilities are innumerable. It is, therefore, no wonder that governments at all levels are trying to infuse the latest information technology (IT) and telecommunications technologies into disaster response and management procedures.

As the emergency responders' reliance on IT-based infrastructure increases, providing the means to access this infrastructure becomes significant in organizing an effective response. However, ensuring this access is almost always a challenge because the existing infrastructure is usually rendered useless or the disaster may take place in an area where there was no infrastructure to begin with. To tackle this problem, researchers at CWT have been collaborating with industry partner Science Applications International Corporation to develop a rapidly deployable wireless communication system for emergency response [3, 4]. For a fully functional communications system, three levels of hierarchy need to be addressed: (i) local connectivity, e.g., using wired and wireless local area network (LAN) technology; (ii) backbone or backhaul connectivity; and (iii) wide area network (WAN) connectivity in the form of the global Internet or a private network [5]. The system focuses on the second level and is intended to provide a 120-Mbps backbone network to link a hub and up to eight remote Disaster Response Gateway (DRG) units (or simply a "Gateway" unit). The hub DRG can use surviving network infrastructure at the periphery of the disaster area or use a satellite earth station to provide a link to the outside world. Alternately, the system can also be used to create localized networks within the disaster area. The field DRG units can provide wired Ethernet or wireless local area network access to portable or laptop computers as well as other network enabled devices like hand-held devices, web cameras or voice-over-IP (VoIP) phones.

2.2.1 Prototype Network

The network topology, shown in Figure 2.1, consists of a base station (or hub) and multiple field Gateway units (or remotes) that are connected to each other by a LMDS wireless backbone. The prototype network that is being deployed will consist of a hub and from two to eight remotes. Each hub and remote Gateway performs network services such as routing. The backbone network is functionally equivalent to a network bridge. For example, consider that the hub is connected by a 10/100-Mbps Ethernet connection to the external WAN network and a remote unit is connected to the end host through a 10/100-Mbps Ethernet connection. Then the LMDS backbone network essentially serves as a virtual Ethernet bridge, i.e., Ethernet packets coming in and Ethernet packets going out of the backbone network.

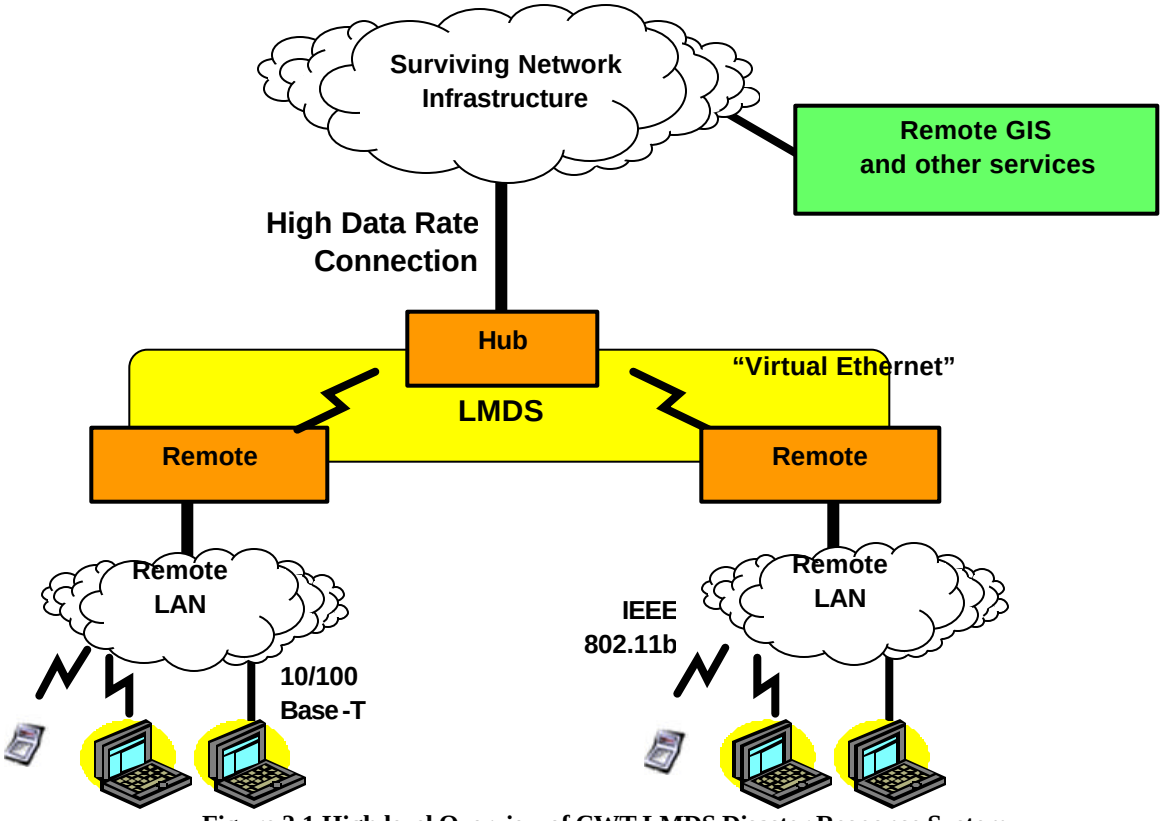


Figure 2.1 High-level Overview of CWT LMDS Disaster Response System

The hub uses a high data rate connection such as 10/100-Mbps Fast Ethernet or a SONET STS-3 connection to access a wide area network. The WAN connection may be made through the surviving terrestrial network infrastructure. In cases of severe damage or absence of any previous network infrastructure, satellite-based network access points can be used. The remotes can be scattered across the disaster area to serve a radius of up to 5 km from the hub. The remotes can provide personal digital assistants (PDA) or laptops, carried by field personnel, with LAN services ranging from 10/100-Mbps Ethernet to IEEE 802.11 wireless connectivity. Thus, end hosts can access the network servers and applications such as Geographic information system (GIS) on the wide area network using the LMDS wireless backbone.

2.2.2 System Components

Besides providing a high data rate "pipe" for deploying new disaster response applications, the system developed at CWT features several innovations that aid the rapid deployment and robust operation of a disaster response communication system. A broadband channel sounder [6] is integrated into the hub and field units to allow measurement of channel characteristics. Information from the sounder can be used to optimize the final placement of the hub and field units. Along with a suite of GIS tools, the sounder can be used to enable the system to be quickly and reliably deployed. Since the network will be used as a communication backbone during disaster situations and since it must maintain communications with possibly varying channel conditions, the network should use an adaptive scheme to improve Transport Control Potocol/Internet Protocol (TCP/IP) performance. The adaptive data link protocol, described in Section 2.2.3, adjusts error coding and error recovery schemes during operation. Sounder information may also be used to adjust link configuration based on observed channel conditions, thus making the system more robust to sub-optimal deployment and a changing environment.

The disaster response system consists of a hub Gateway and multiple remote Gateway units. The hub and remote units are identical except for the way they are programmed. The hub units are programmed to transmit during all time slots whereas the remote units are programmed to follow the multiple-access scheme.

Each unit, as shown in Figure 2.2, contains the following subsystems.

1. Gateway Controller subsystem: The gateway controller subsystem forms the core of the disaster response system. The subsystem consists of the LMDS Gateway Controller and three other modules – Quadrature Phase Shift Key (QPSK) Modulator, QPSK Demodulator-Digital, and QPSK Demodulator-Analog. The QPSK Modulator and the two QPSK Demodulators are collectively referred to as the QPSK Modem. The QPSK modem is a commercial satellite modem that has been adapted to use for terrestrial LMDS. The LMDS Gateway Controller design and implementation forms the focus of this work.

2. Radio subsystem: The radio subsystem consists of the 28-GHz LMDS band radios and antenna components. The LMDS radios up-convert the intermediate frequency (IF) output (400 MHz) of the QPSK modems to the LMDS band for transmission.

3. Sounder subsystem: The Sampling Swept Time Delay Short Pulse (SSTDSP) Sounder (or Sounder) is a novel channel measurement tool that can be used to profile the channel performance in real time [6].

4. Host computer: The Host computer provides an interface to the Gateway Controller and Sounder. It contains GIS and Sounder control software that can aid in reliable and rapid deployment of the remote units. The Host computer also interfaces to the Gateway Controller using a standard serial interface that allows the Gateway Controller monitor software to modify system parameters,

such as FEC levels and ARQ, based on Sounder data. The Gateway Controller monitor software can also be used to obtain statistics on the error rates, data rates, etc. from the Gateway Controller through a formalized command set.

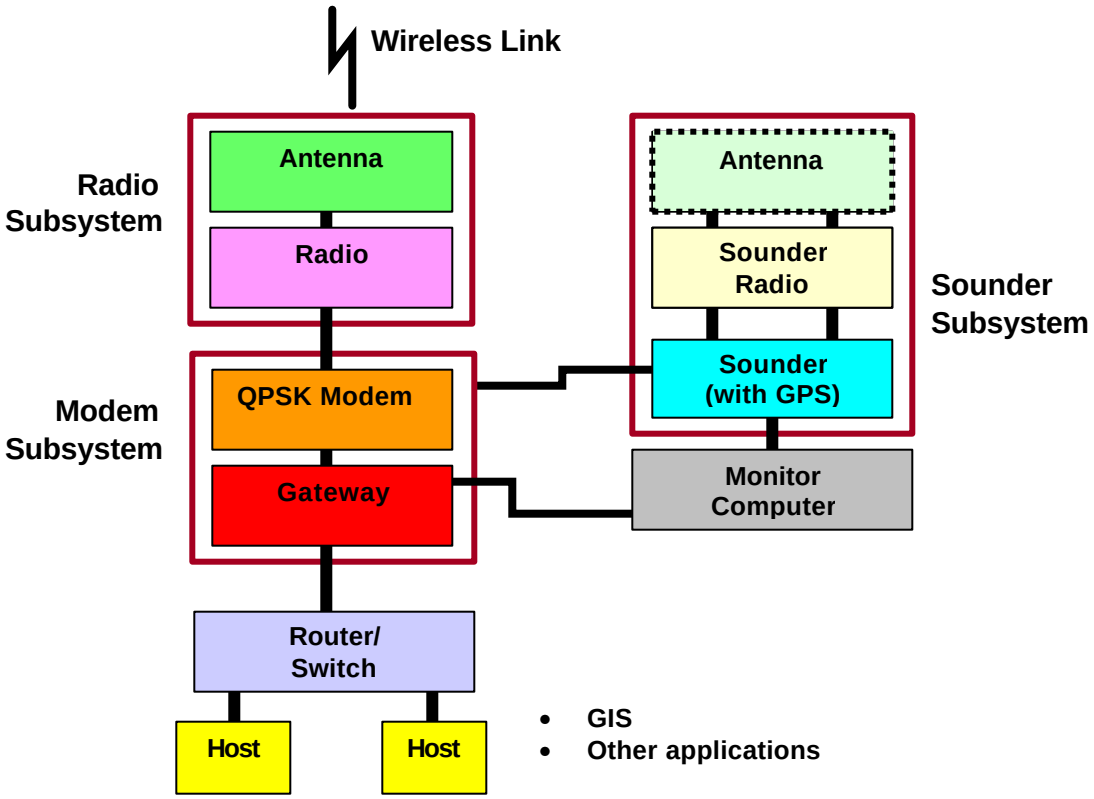


Figure 2.2 CWT LMDS Disaster Response System: Hub/Remote Unit Components

2.2.3 Multiple Access Scheme

The network supports bi-directional traffic between the hub Gateway and the remote units. The system uses a TDMA – FDM (time division multiple access – frequency division multiplexing) scheme, to allow full duplex connection between a hub and up to eight remotes. The hub broadcasts its transmissions on the "downlink" frequency (f_d) so that all remotes receive the same transmission at the same time. The

remotes can transmit to the hub on the "uplink" frequency (f_u) based on a TDMA scheme [7]. Each remote is assigned a statically allotted time slot in which it can transmit data. The frequency and direction of transmissions between the hub and remotes are shown in Figure 2.3.

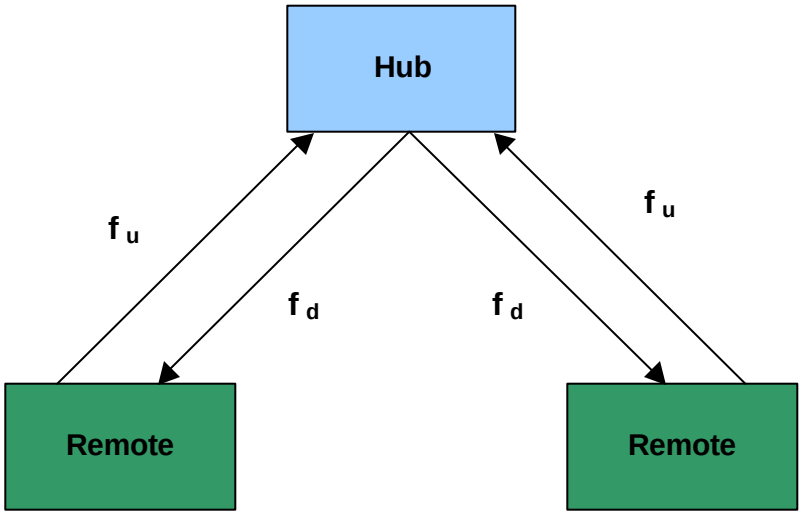
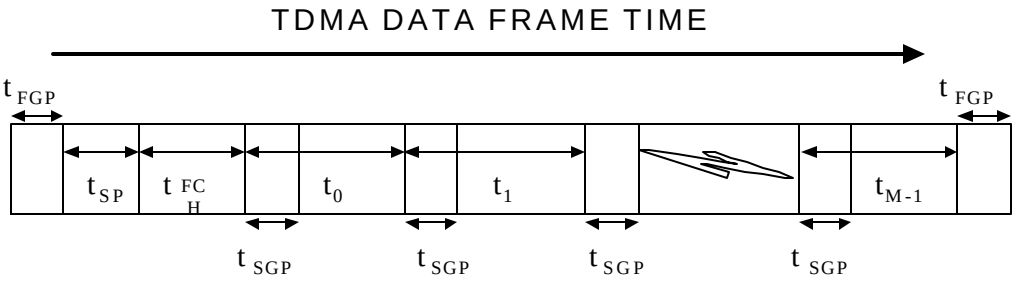


Figure 2.3 Frequency Division for Hub and Remote Transmission Units

The TDMA multiple access scheme [7], developed at CWT, allocates the transmission time slots to the hub and remotes and allows for the integration of other system components such as the sounder. A "Data Frame" consisting of time slot divisions is shown in Figure 2.4 and the "Super Frame" consisting of 512 Data Frames and a Sounder operating time is shown in Figure 2.5.



t_{FGP}	Frame Guard Period	t_{FCH}	Frame Control Header
t_{SGP}	Slot Guard Period		
t_{SP}	Synchronization Period	$t_0, t_1, .., t_{M-1}$	Data Slot

Figure 2.4 TDMA Data Frame Format [7]

t_S	Sounder Frame
t_D	512 Data Frames
t_{SP}	Synchronization Period
t_{FGP}	Frame Guard Period

Figure 2.5 TDMA Super Frame Format [7]

Each data frame consists of *n* slots where *n* is the number of remotes. Each slot is separated by a guard time called slot guard time to prevent overlaps in transmission. Each frame begins with a synchronization period that allows the hub and remotes to maintain relative synchronization. A Frame Start preamble transmitted by the hub during the synchronization period allows the transmission time slots to align identically at each receiving remote site. Similarly, each Data frame is separated by a frame guard period. The Super Frame consists of a synchronization period to allow the sounder to operate for a fixed time known as Sounder Frame. During this interval the Sounder will assess the quality of the radio transmissions that carry data. This assessment is used by the hub and remotes to adjust the transmission rates, coding levels and retransmission characteristics of the system. The multiple access scheme is explained in detail in [7].

2.3 Informal Specifications Summary for the LMDS Gateway Controller

The system architecture description forms the basis for defining a set of informal specifications, including functional, performance, cost and architectural aspects, for the Gateway Controller design. The disaster response system is designed to provide an effective bandwidth of up to 10 Mbps per user for eight users distributed over a disaster area of radius up to five kilometers [7]. The aim is to provide high data throughput associated with providing network connectivity and multimedia applications.

The disaster response system will be used to respond to natural and man-made disasters where the unknown nature of the environment and adverse weather conditions can lead to high bit error rates. These high error rates can hurt performance, especially for TCP because this protocol responds to loss due to congestion in the same manner as it responds to loss to due to error. Adaptive protocols that support variable FEC and optional ARQ schemes can improve TCP/IP performance in such situations and should be employed in the Gateway Controller design.

Since the disaster response system is designed to take advantage of existing network infrastructure whenever possible, the network interfaces must be chosen

carefully. The system is to use 10/100-Mbps Ethernet network interfaces due to widespread popularity and ubiquity of the Ethernet standard.

Functionally, the Gateway Controller encapsulates Ethernet packets with the LMDS MAC protocol described in Section 2.2.2 on one end and then translates them back to Ethernet packets at the other end of the LMDS wireless link. The LMDS Gateway Controller performs the following functions.

1. Provide Power, Signal and Control interfaces to the QPSK Modulator, QPSK Demodulator-Analog, and QPSK Demodulator- Digital modules as required.
2. Provide Physical and Data link layer functions for Ethernet Interface
3. Implement the TDMA scheme for multiple remotes to share the medium in the uplink frequency. Maintain timing and synchronization between hub and remotes at the bit, packet and frame levels.
4. Implement Link Layer retransmission and Adaptive FEC to reduce network delay due to bit errors
5. Interface to Network Monitoring/Control and Radio Monitoring/Control Software in the host computer and the Sounder.

The Fast Ethernet interface at 100 Mbps and the QPSK wireless interface at 120 Mbps also place strict processing constraints and result in reduced delay tolerances down to the order of microseconds. We believe that these speeds and tolerances are achievable by careful component selection, design and programming.

The Gateway Controller is essentially developed in a research environment, and uses experimental protocols that aim to improve TCP/IP performance over wireless channels. Therefore, the architecture must be flexible enough to accommodate late protocol changes and modifications. Also, as a result of academic research environment, cost is always an important constraint and at times may be an over-riding one.

2.4 Summary

To meet the functional requirements the Gateway Controller must, in turn, satisfy the high performance specifications needed for the signal processing and forward error correction. To achieve high-performance and increased flexibility for making protocol changes the system architecture should essentially use programmable and configurable components. Chapter 3 deals with the architectural issues in more detail.

CHAPTER 3

Design Methodology for Wireless Embedded Systems

3.1 Overview

The scope of our design methodology extends from specification to implementation. The discussion of the application system in the previous chapter fixes the application or service requirements; i.e., it determines the functions, speeds, power requirements, form factor, etc., that are required by the applications or higher layers of the protocol stack. Once the medium access and link level protocols are defined, the next step is to implement them in hardware and software.

3.2 Relating Abstract Specifications to Implementation

Protocol specification must define the services, behavior and formal sequences of message exchanges between communicating nodes or layers. The protocol definition must define behavior for all possible situations and circumstances. While this in itself can be complex, the implementation phase of a protocol poses additional challenges of its own. The complexity of the design process is significant in an integrated design approach, such as in the case of MAC protocols. MAC implementation solutions consist of a mix of hardware and software since they require close interactions with the underlying physical layer and require quick responses to events. To improve reaction times and power efficiency, it is highly desirable to implement the control logic in hardware. On the other hand, easier product upgrades and higher flexibility of a software-based approach favors implementing as much as possible in software. In practice, most implementations consist of a mix of both hardware and software. The challenge of designing these hybrid systems is referred to as “hardware-software co-design”. The challenge is that hardware and software have inherently different design styles,

representation and testing techniques. The hardware-software co-design challenges are explored in the following sections.

3.3 System-level Design Methodologies

Hardware-software co-design requires a more comprehensive and cohesive design process including both hardware and software in the design to overcome various design challenges. Moreover, designers must also contend with rapidly changing or evolving standards and specifications. Support for late protocol changes requires that the target implementation have enough flexibility to incorporate future design or algorithmic changes. The increasing importance that is being given to energy considerations is another factor to be dealt with in making architectural choices. All of the above requirements necessitate a flexible, low energy, high-speed architecture and a well-understood general system level design methodology upon which novel communication systems can be built.

3.3.1 Platform-based Design

The emergence of a number of wireless standards like Bluetooth, IEEE 802.11 and IEEE 802.16 has created a market for numerous wireless applications and products. However, the rapid emergence of protocols and their successive variations, as in the case of IEEE 802.11, have decreased the time-to-market budgets even as the useful lifetime of these products are rapidly decreasing. One of the solutions to reduce design times suggested by Ferrari, et al. [8] and Kuetzer, et al. [9] is platform-based design by "orthogonalization" or separation of design space concerns. Platform-based design aims to reduce design time by facilitating reuse using abstractions called "platforms." One example of a platform, called a hardware platform, consists of a set of parameterizable architectures that satisfy the constraints and support the functional specification of a design. Similarly, a software platform is a software layer consisting of the real-time operating system (RTOS) and device drivers that allows for abstraction of the hardware platform through an interface called the Application Program Interface (API). The

combination of the hardware and software platforms constitutes the system platform. Platform-based design helps designers to swiftly design prototypes by re-using readily available and tested components from a library, which in this case is called a platform. Component re-use will not only significantly reduce design time and effort but also helps to reduce time invested in testing those modules. The disadvantage of platform-based design is that it may limit the designer to a smaller design space provided by the platform.

3.3.2 Platform-based Design Terminology

"Platform conception" is the process of developing hardware and software platforms. It is imperative that the target application be fully understood before embarking on a platform design. The first step in the development of a platform, *functional profiling*, is to identify and extract common functionality and features of the application domain. The next step, *architecture exploration*, is to identify architectures for these functions that would deliver adequate performance while satisfying the constraints of the application domain. "Platform instantiation" involves mapping functionality onto specific system modules that result in optimal performance. The *mapping process* involves the selection of an optimal architecture among the various architectures determined to be suitable and identifies components that can adequately satisfy the performance requirements. When all the design constraints are satisfied, the *implementation* of an application becomes software based. The application designers only need to focus their attention on the application software compilation and hardware synthesis to create an application. All of the steps in the platform-based design flow are explained in detail in the following sections.

3.3.3 Platform-based Design Methodology

The platform-based design methodology can be broadly classified into three phases, platform conception, platform instantiation and implementation [10]. The platform-based design flow showing the major stages is represented in Figure 3.1.

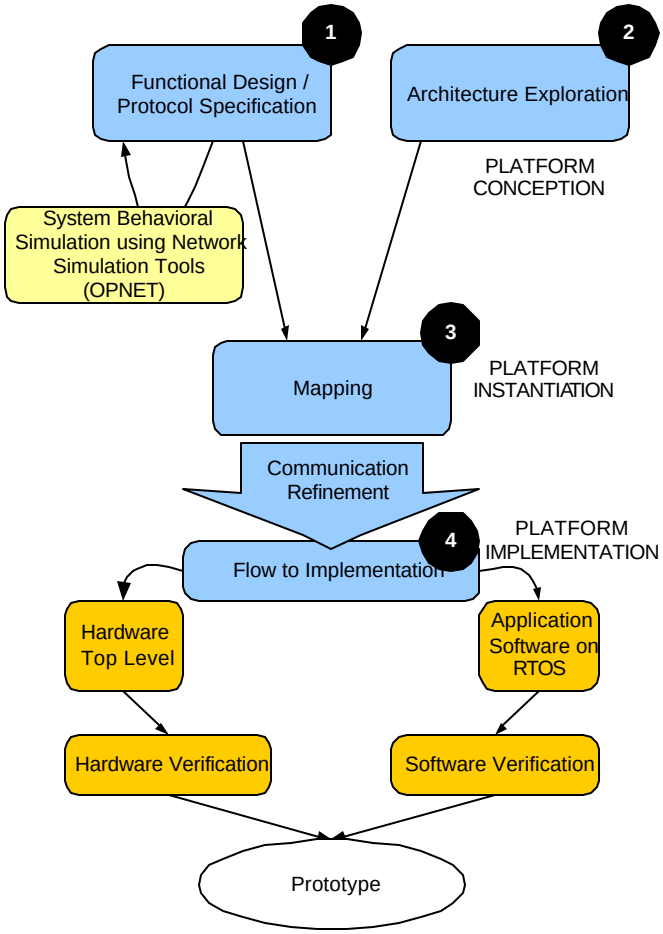


Figure 3.1 Platform-based system design methodology

3.4 Configurable Platform Design Flow for Gateway Controller

3.4.1 Functional Profiling of Gateway Controller

Functions in communication systems can be broadly classified into data processing and control functions [11]. Most communication systems perform "packet processing" parallel operations as well as bit-serial data processing operations. Both function types can be further classified as operations that modify data and those that merely transport them. Control functions include event processing and decision making functions. For the design being performed here, these functions are represented as finite state machines that can be mapped to either hardware or software. The functions identified are helpful in understanding the complexity and the requirements for the system.

The functional blocks that form the transmit path and the receive path of the Gateway Controller are shown in Figures 3.2 and 3.3, respectively. The transmit path for the Gateway Controller refers to data path from the Ethernet PHY receiver to the QPSK modulator and radios. Similarly, the receive path refers to the data path from the radio and QPSK demodulator to the Ethernet PHY transmitter.

The Fast Ethernet PHY and MAC in the transmit path implement the Carrier Sense Multiple Access /Collision Detection (CSMA/CD) algorithm specified in IEEE 802.3 standard. The Ethernet packets can vary in size from 64 to 1500 bytes. The LMDS MAC payload builder uses the Ethernet segmentation block to break the Ethernet packets into sizes that are suitable for the LMDS MAC payload. The payload is then encoded using Reed-Solomon (RS) FEC code at the appropriate encoding level. Once the payload has been built, the LMDS MAC header builder can form the header. The header is then encoded at a standard FEC level by the RS header encoder. Finally the payload is encoded using the Turbo Product Code (TPC) encoder. The encoded data stream is then formatted by a symbol-mapper for transmission using a QPSK modulator and radios. The preamble generator is used to generate the bit patterns that denote the beginning of the

frame and each data transmission. The transmit path also needs to maintain a timer to schedule the order of transmissions between multiple remote Gateway Controller units.

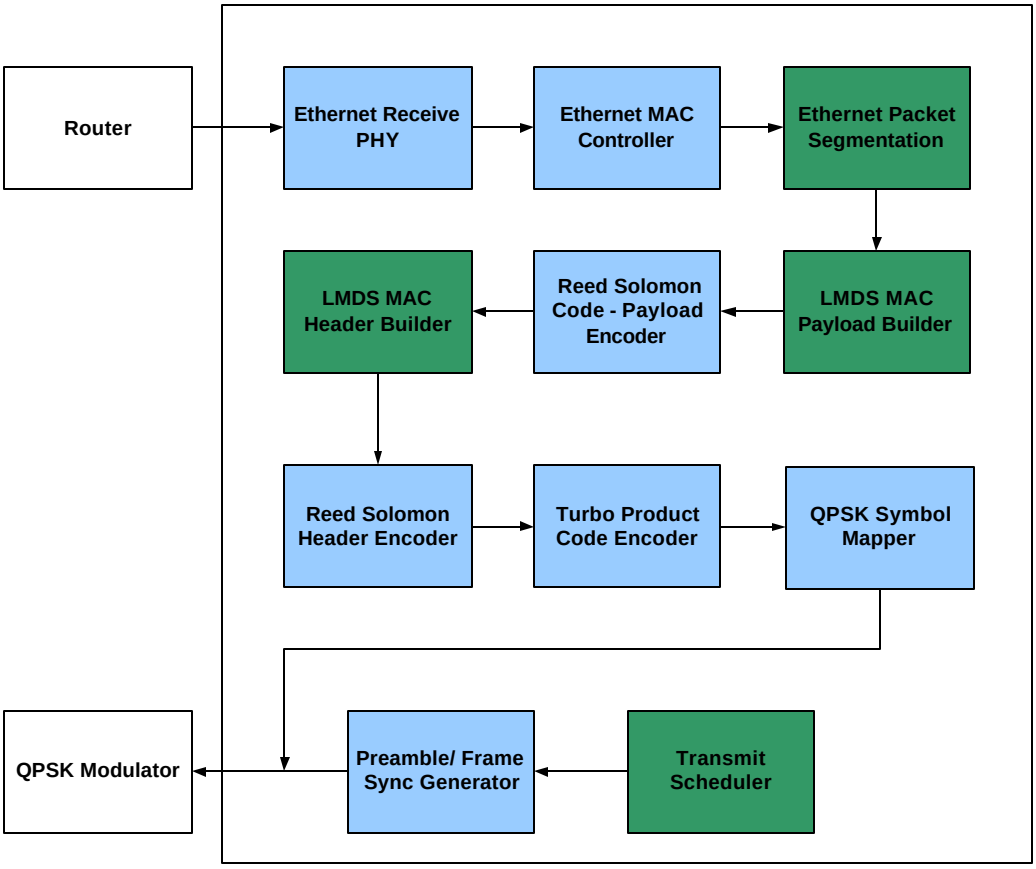


Figure 3.2 Functional blocks in transmit path of the Gateway Controller

The receive path, shown in block diagram of Figure 3.3, performs the same functions as the transmit path except in reverse order. The QPSK symbol-demapper reformats the QPSK symbols into a bit-stream, which is then decoded by the TPC decoder. To obtain information on the RS coding levels of the payload, the RS header decoder must first decode the header. The LMDS MAC packet header is then analyzed. The MAC packet header decoder gleans information regarding coding levels, retransmissions and acknowledgements from the header. Once the coding levels are known the RS payload decoder decodes the payload to obtain the transmitted data from

the encoded packetized bit-stream. The Ethernet re-assembly block reconstructs the segmented Ethernet packets for transmission by the Ethernet MAC and PHY. The preamble detector serves to identify bit patterns transmitted that signify events such as the beginning of the data frame and super frame. See Section 2.2.3 for an explanation of frame characteristics used here.

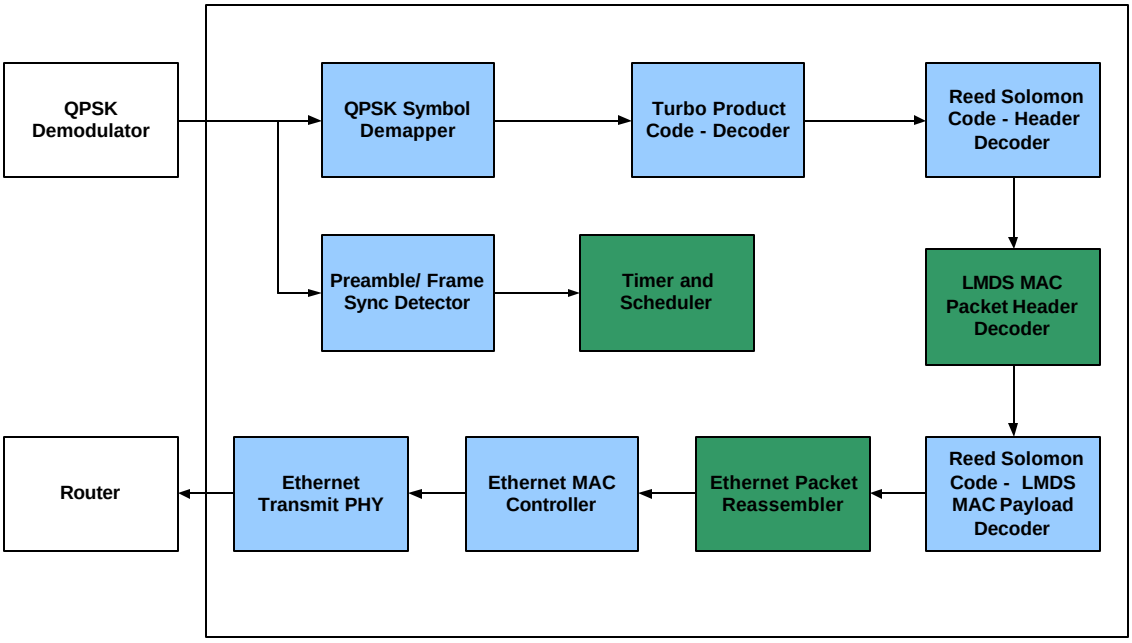


Figure 3.3 Functional blocks on the receive path of the Gateway Controller

Having identified the various functions that are expected of the design we can move ahead to survey architectures that would result in efficient implementations of these functions. The fact that there are functions such as FEC which tend to be cycle-intensive and inefficient in terms of power when implemented in software suggest that the architecture should have a good balance of hardware and software.

3.4.2 Architectural Exploration

Before considering any set of architecture topologies, the design space for the application domain must be identified and bounded. In this case the design space includes

general and specialized microprocessors, digital signal processors, programmable logic devices and custom application specific integrated circuits. For a platform-based design methodology to be robust, it must be able to adjust to application redesigns and improvements without much change to the base platform elements. Therefore, the platform must be built on a foundation of configurable architectures and parameterizable elements that are flexible enough to allow for easy integration and scaling.

The simplest and most common architecture used in traditional communication system designs is to configure a microprocessor or DSP with a set of application specific peripherals. In this scenario all the system blocks are mapped to software running on the processor or DSP. Though this provides a lot of flexibility, the serial processing model of software-based design limits system performance to a great extent. However, ASICs yield very high performance, but require significant non-recurring engineering (NRE) cost and effort. ASICs offer low flexibility to the designer and require tremendous redesign efforts in the face of changing specifications or standard updates. This makes the ASIC option unsuitable for the development of products that are based on standards that have not yet stabilized or are undergoing development – a trend that is common in today's industry. Given the above requirements programmable logic devices like FPGAs offer an excellent alternative. High density FPGAs and readily available configurable IP cores provide significant performance improvement by allowing designers to take advantage of parallelism and pipelining processing stages.

Designers can also take advantage of hybrid architectures involving a processor and a FPGA co-processor to achieve almost triple performance for certain "cycle-intensive" operations [12]. The choice of the processor is crucial and determines whether a system block is implemented as hardware or software. The processor choice also determines system parameters like voltage levels, input/output (I/O) standards and most importantly the system bus protocol. The new breed of specialized network and communication SoC processors are ideally suited for control and some "packet processing" operations in high bandwidth applications and have built-in network interfaces that simplify design.

Once the microprocessor target is identified, the software platform, which interfaces the hardware to the programmer through device drivers and the Application Program Interface (API), can be developed. RTOS selection and multiple operating system support for the target hardware platform must be weighed before developing the software platform. Driver support for specific devices included in the hardware configuration can strongly influence RTOS selection.

3.4.3 Mapping

Various iterations (Designs A, B and C) produced during the design process, were considered for the configurable hardware platform. Their suitability to the application domain was studied with an eye toward optimizing the parameter mentioned earlier in this report.

Design A consisted of a single general-purpose microprocessor with all functionality mapped into software. Even initial analysis of the design suggested that the system would not be able to handle high bandwidth data processing requirements. The major problem was the processor bus-bottleneck and the serial processing model. A series of operational speed calculations, based on possible process clock speeds and instruction execution times, were performed. Design A was not analyzed in detail further, even though it may have been suitable for lower bandwidth applications, as our goal was to identify designs that satisfied high bandwidth requirements.

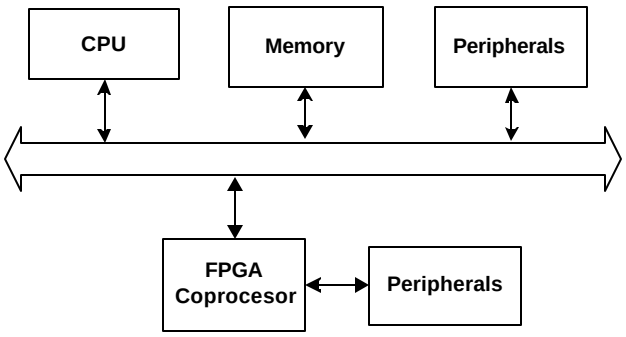


Figure 3.4 Design B: General purpose processor and FPGA Co-processor

Design B, shown in Figure 3.4, consisted of a general-purpose processor along with a FPGA co-processor. Various processors, like the i960Jx and PowerPC family, and bus protocols were considered. The FPGA co-processor, implemented on a Xilinx Virtex FPGA, acted like a DMA controller by moving data and reducing the processor load. The control operations are implemented in software running on the processor. Some bit-serial operations, like the modulation symbol mapping and radio interface, are moved to hardware logic on the FPGA. This simplifies the serial interfaces and also reduces bus bottlenecks.

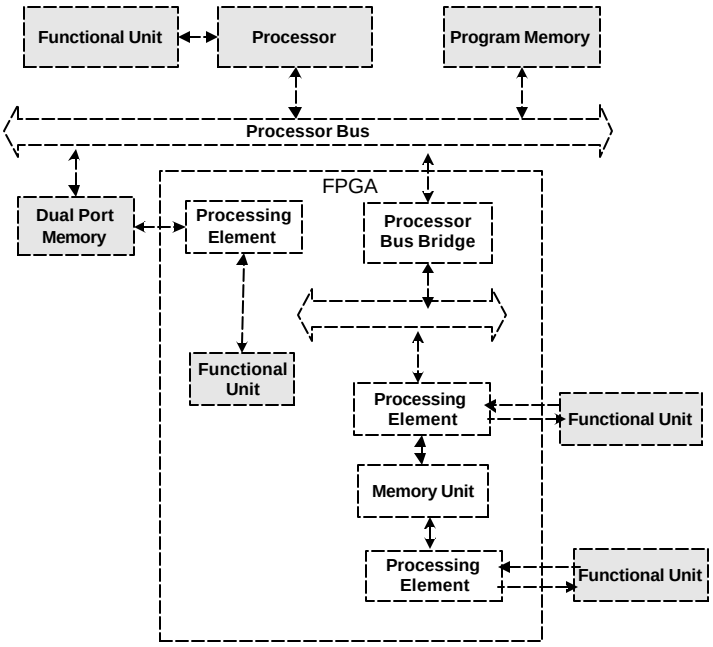


Figure 3.5 Design C: featuring a memory-centric architecture

The third iteration, Design C, shown in Figure 3.5, was proposed to take greater advantage of IP core reuse and further simplifies interfaces to promote easier integration of heterogeneous system blocks. To achieve this objective, a "memory-centric" architecture with a combination of a specialized processor for packet-level operations and programmable logic devices like FPGAs for bit-level operations was implemented. High-density FPGAs and readily available configurable IP-cores provide significant performance improvement by allowing designers to take advantage of parallelism and pipelining processing stages.

The architectural elements can be broadly classified as control or data path elements [15]. The control elements deal with timing, status and ordering functions. These elements are implemented in the processor. The data path elements consist of functions that move, alter or add to the data that is transmitted or received. The data path elements can be further classified into:

- "Functional Units" (FU) which modify or transform data along the data path,
- multiple scattered DMA-like "Processing Elements" (PE) that transport or relocate data, and
- "Memory Units" (MU) to store data between stages of the pipeline.

The functional units perform various phases of the data processing before transmission and after reception. Forward error correction CODECs or modulation symbol-mappers are examples of functional units. The processing elements perform data transfers between data processing stages or functional units. The architecture is designed to be memory centric in that distributed memory elements like dual-port synchronous Random Access Memories (SRAM) exist between each functional stage of the data path. This modular architecture permits designers to take greater advantage of IP core reuse and further simplifies interfaces to promote easier integration of heterogeneous system blocks.

A dual-port memory, accessible by both the microprocessor and the FPGA-based processing elements, allows the processor and FPGA logic to read and write data simultaneously. Further, embedded dual-port SRAM memory available within the FPGA is used to create distributed buffers between various processing stages. Most control functions are implemented in software while some critical functions, like timing references, are mapped to the FPGA blocks. Also, to take advantage of the optimizations in the specialized processor, the packet processing functions are performed in software. Among the many communication and network processors considered, the Motorola PowerQuicc II SOC platform was found to be most suitable for the application. The Motorola PowerQuicc II 8255 communication processor [13] consists of a high performance 64-bit, 200-MHz PowerPC core and a 32-bit, 133-MHz communication processor that simplifies network interfaces with support for Fast Ethernet, ATM and T1/HDLC protocols. The Xilinx Virtex XCV600 FPGA [14] offers high gate densities (up to 1M+ gates), 512 I/Os and up to 16 KB of internal single/dual port SRAM embedded memory.

3.5 Summary

The Gateway Controller design implements a TDMA MAC scheme with different uplink and downlink frequencies on the wireless link and an Ethernet interface on the wire line side. Given the high bandwidth required and the amount of data that needs to be transported between blocks, the hybrid architecture of Design C was used with processing elements implemented partly as software running on the communication processor and partly as FPGA hardware logic blocks. The hardware platform is discussed in detail in Section 4.2. The software platform consists of a board support package (BSP) developed for VxWorks RTOS from Wind River Systems and device drivers for Ethernet, Universal Asynchronous Receive/Transmit (UART) and other common functions. This can then be used to generate and compile the application software. The software platform is discussed in detail in Section 4.3.

CHAPTER 4

Gateway Controller Implementation

4.1 Overview

This chapter presents, in detail, the implementation of the data processing modules of the LMDS Gateway Controller into physical hardware and software components. The chapter is included for completeness and to explain how the data processing modules relate to the wireless link protocol. The various design choices and performance tradeoffs are also described.

4.2 Gateway Controller Hardware Implementation

Though the data processing modules were designed to operate autonomously, the microprocessor was retained in the implementation primarily for the flexibility afforded by the processor. Including the processor makes exploration of various networking protocols easier, as it allows the exploration to be done in software. The processor functionality is particularly suited to performing the following tasks: packet transmission scheduling over radio link and support for medium access protocols, including packet header processing. The higher-level MAC packet processing functions are more efficiently implemented as processor micro-code. However, some bit-level tasks of the physical layer, that can be performed by the processor are implemented in the FPGA as they offer more efficient processing ability within affordable silicon area. The FPGA-based interface to the radios allows flexibility to enable greater experimentation with physical-layer protocols and take advantage of new radios as they become available. The combination of the processor and FPGA logic blocks yields the most efficient implementation that accomplishes the tasks at high enough speeds.

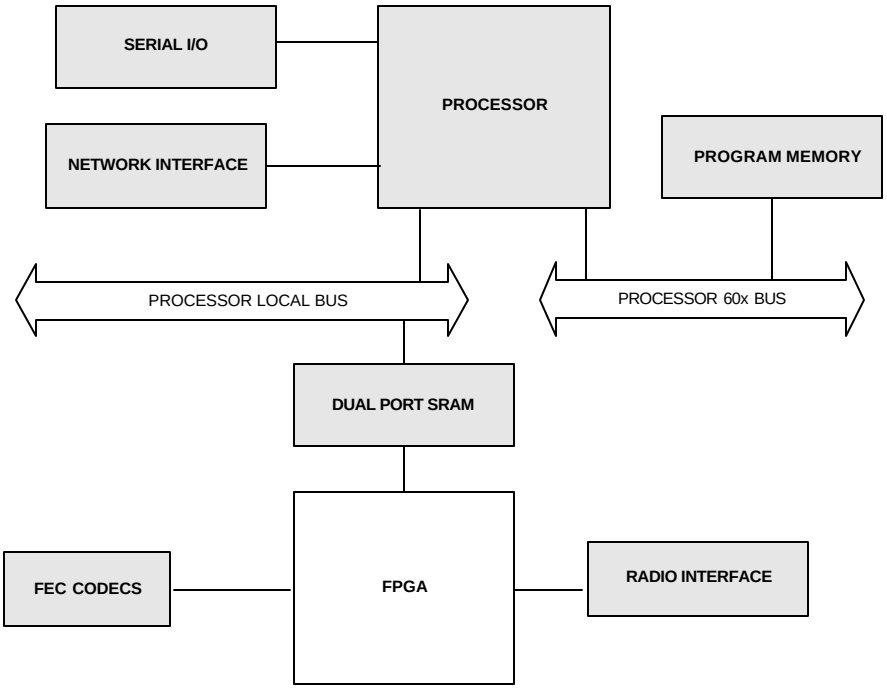


Figure 4.1 Gateway Hardware Computational Resources

4.2.1 Microprocessor Subsystem

The microprocessor is responsible for system initialization and various high-level protocols. As shown in Figure 4.1, the system is controlled by a Motorola MPC8255 PowerQuicc II communications platform [13]. To perform its varied functions, the SoC platform shown in Figure 4.2 integrates a PowerPC 603e reduced instruction set computer (RISC) microprocessor running at 200 MHz, a communications processor module (CPM) running at 166 MHz, with 16 MB of read/write SDRAM, and 8 MB of Flash memory for program storage. The processor micro-code implements the TDMA MAC scheme for the LMDS network and closely controls the FPGA logic that implements the lower layer functions and interfaces.

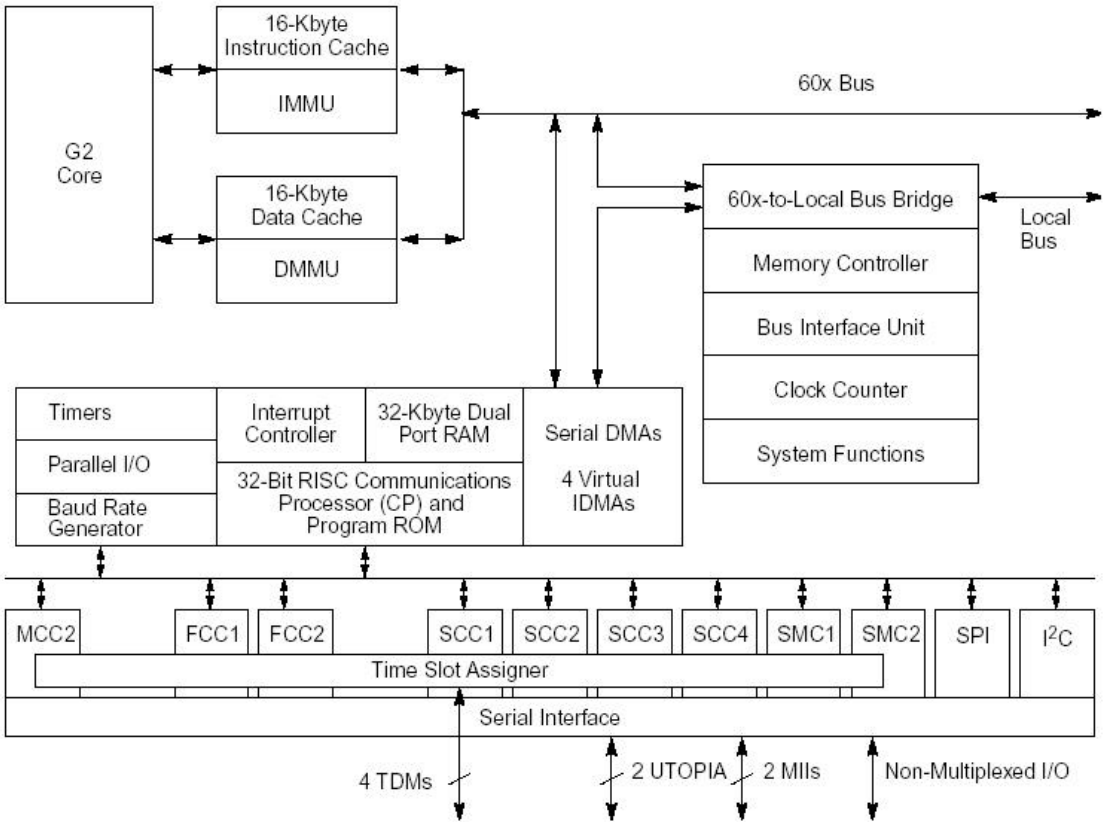


Figure 4.2 Motorola MPC8255 Power Quicc II Communications Platform Block Diagram [13]

The MPC8255 has a built-in Ethernet MAC controller in its Fast Communication Controller (FCC) and connects to an external Ethernet PHY device through the Media Independent Interface (MII). The MPC8255 also contains a UART interface, which can be connected to an RS232 interface chip.

4.2.2 FPGA Co-processor

The FPGA is a XCV 600 FPGA in a 680-pin fine-pitch ball grid array (FBGA) package that belongs to the Xilinx Virtex family of FPGAs. The Virtex-family FPGAs offer a wide variety of programmable system features; a rich hierarchy of fast, flexible interconnects; and advanced process technology. Virtex family architecture, shown in Figure 4.3, delivers high-speed and high-capacity programmable logic solutions that enhance design flexibility while reducing time-to-market [14]. Virtex function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function

generator, each LUT can provide a 16 × 1-bit synchronous RAM. Virtex FPGAs also incorporate several embedded large block RAM (BRAM) memories, which complement the distributed LUT-based memories implemented in combination logic blocks (CLB) that provide shallow RAM structures. Block RAM memory blocks are organized in columns. All Virtex devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Virtex device 64 CLBs high contains 16 memory blocks per column, and a total of 32 blocks. Another attractive feature of the Virtex family is it's high I/O pin counts and configurable I/O buffers (IOB) that can be programmed into a wide variety of standards. This allows the FPGA to interface with multiple IO standards like Low-voltage TTL, 5V TTL, CMOS and PECL.

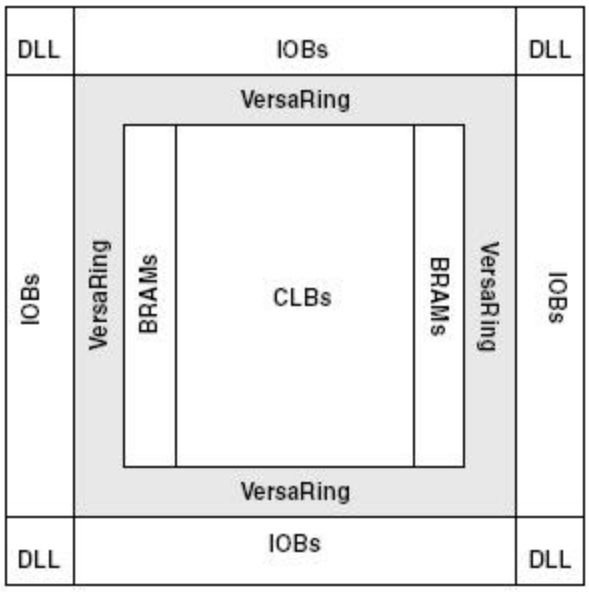


Figure 4.3 Virtex FPGA Family Architecture

The FPGA co-processor architecture process flow is shown in Figure 4.4 and the processes can be broadly grouped under Transmit or Receive process flows. The transmit process flow, shown by a dotted red line in Figure 4.4, refers to the data path from the Ethernet PHY receiver to the QPSK modulator. The receive process flow, shown by a

solid blue line in Figure 4.4, refers to the data path from the QPSK Demodulators to the Ethernet PHY transmitter.

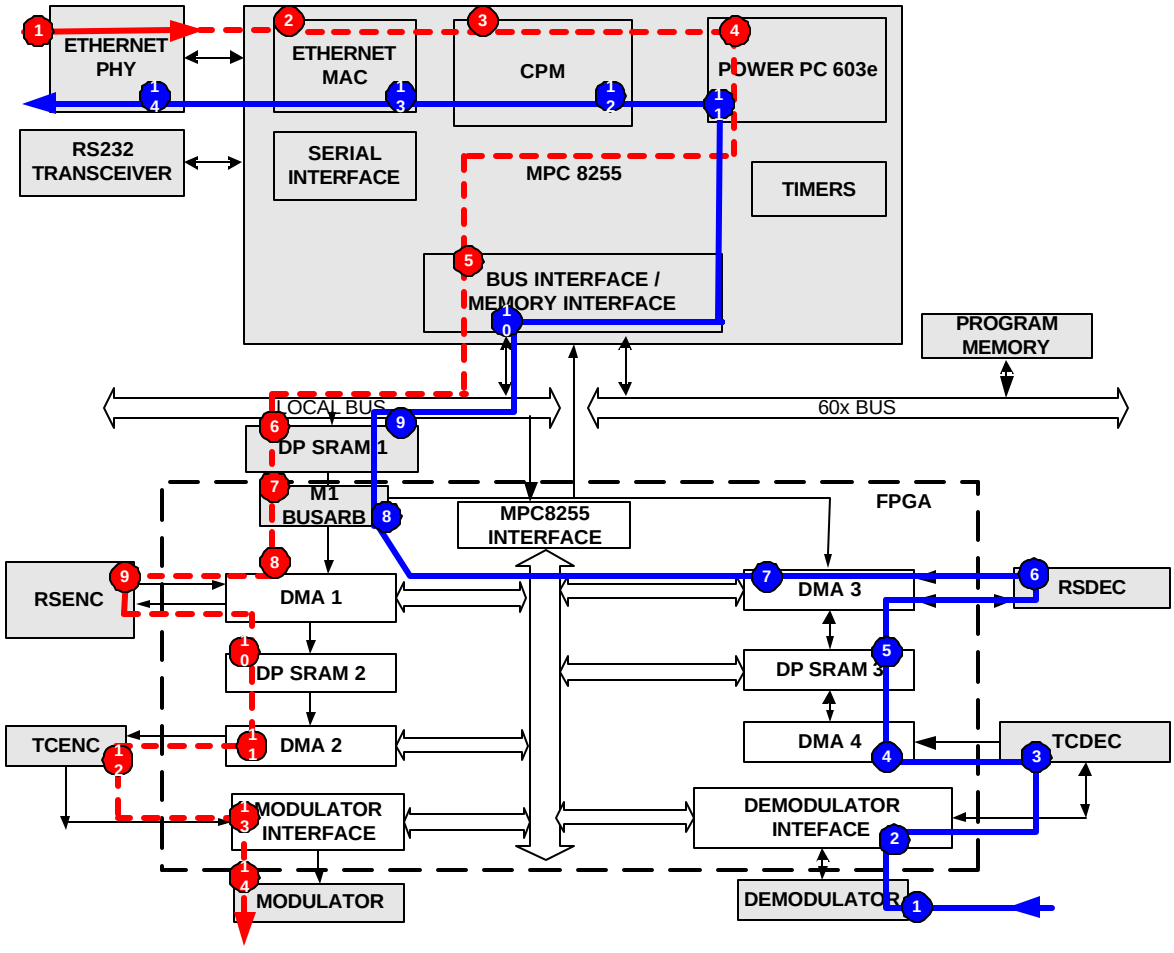


Figure 4.4 FPGA logic - process flow overview

4.2.2.1 Transmit Process Flow Overview

The processor reads in data from the Ethernet PHY chip and stores it in the external dual-port SRAM memory. Using this data, the processor then builds the wireless MAC payload after segmentation of the Ethernet packets to the appropriate size, and again stores it in the external dual-port memory. All processor accesses to the memory

are through the one port referred to as the “right” port whereas all FPGA accesses are through the second port and referred to as the “left” port.

The processor next builds the header in the external DP SRAM, after looking for updates/acknowledgements to add to the header from the receive process. Depending on whether the acknowledgement field sequence number requested by the receiver transmission of the new payload or a retransmission of the unacknowledged payload is scheduled. The updates in the header maybe acknowledgements of received payloads on the receive path, requests for retransmissions etc. The processor then passes (writes values to a register through IO ports) the beginning address in external DP SRAM and the RS coding rate of the payload packet to the first DMA module in the FPGA. After that it asserts the DMA1 start (DMA1_START) signal. The DMA1 accesses the payload in external DP SRAM through the left port after requesting access to the arbiter in the FPGA. The DMA1 process is explained in detail in Appendix B. The arbiter grants the request if the left port is not being used by the receive process. If the left port is busy then the arbiter waits till the receive process relinquishes control of the left port. Once DMA1 has completed writing the payload to the internal transmit memory, it then asserts its DONE signal (DMA1_DONE).

When the processor receives an interrupt indicating DMA1 has completed, it can then initiate the DMA2 process. The DMA2 process is explained in detail in Appendix B. All DMAs have at least a START input and DONE output signal. This is done mainly to let the processor keep track of the process flow. The processor also needs to maintain a timer, for it to determine the beginning and end of the transmission time slot. It will maintain timers to control the beginning of preamble transmission and data transmission. The modulator interface begins preamble transmission when it receives the Preamble Transmit signal (TX_PRE) and transmits the preamble until the Data Transmit signal (TX_DATA) is received and the Data (in the Turbo Encoder’s output buffer) is ready for transmission.

4.2.2.2 Receive Process Flow Overview

The demodulator "accepts" data only when the DMA3's LISTEN signal is asserted by the processor. The in-phase (I) and quadrature (Q) symbols from the QPSK Demodulator are fed directly to the TPC Codec through the Demodulator Interface. The TPC Codec is capable of identifying the beginning and end of frame by using the Frame Sync patterns inserted at the time of encoding at the transmitter. The Turbo Decoded Output is moved to the Receive side internal DP SRAM by DMA4.

The header, which is always coded at a known constant rate, is decoded first to determine payload-coding levels. Then the rest of the payload can be decoded. The DMA3 process transfers data from the Receive side internal DP SRAM to the RS Decoder after the RS decoder has been programmed. The output of the RS decoder is written into the external DP SRAM through the left port after requesting access fom the arbiter. The processor can then access the stored payload and perform Ethernet packet re-assembly after which the Ethernet driver can send the Ethernet packet to the PHY chip through the MII interface. The Host computer issues commands and receives status messages through the UART of the processor based on a modem command set defined for this purpose. The Receive process elements are explained in detail in Appendix B.

4.2.3 Forward Error Correction CODECs

The Gateway implements adaptive FEC to improve the throughput of good data by the channel. The gateway MAC protocol uses a combination of Reed-Solomon and Turbo Product Code FEC CODECs. Software implementations of FEC CODECs are highly cycle intensive, i.e. they consume a lot of processor cycles. As a result, the processor may not be available for other more critical tasks and hence they are not suitable for implementation in software. Two options were considered for the implementation of the CODECs in hardware. The first option was to implement the CODECs as FPGA cores. However, during the early development period of the gateway

there were no suitable commercial cores available for the Turbo product codes that matched our performance requirements, though there were commercial cores available for the Reed Solomon CODECs. The second option was to use commercially available ASICs. In the end, the second option was chosen to implement both CODECs to avoid any reliability issues with unproven IP cores and, also, due to the lower cost of the commercial ASICs.

4.2.3.1 Reed-Solomon CODECs

Reed Solomon (RS) codes are a subset of Bose-Chaudhuri-Hochquenghem (BCH) codes and are linear block codes. A Reed-Solomon code is specified as *RS(n,k)* with *s*-bit symbols. This means that the encoder takes *k* data symbols of *s* bits each and adds parity symbols to make an *n* symbol codeword. There are *n-k* parity symbols of s bits each. A Reed-Solomon decoder can correct up to *t* symbols that contain errors in a codeword, where $2t = n-k$.

The Reed Solomon codes are implemented using an integrated RS encoder-decoder solution on a single chip [16]. The RS CODEC contains both a high data rate programmable Reed-Solomon encoder and a separate decoder that will provide Reed-Solomon forward error correction encoding of blocks of eight bit symbols. The Gateway system uses the CODEC to implement adaptive FEC by switching between fixed RS coding levels. The CODEC can be programmed to implement the following RS coding levels though other codes can also be supported: RS(10,18), RS(200,188), RS(200,192), RS(200,196) and RS (200,180). The coding levels are selected based on simulation results from [7]. The encoder and decoder units operate independently and each can be programmed on the fly to select the desired coding level.

The decoder can operate independently to process blocks of up to 255 eight-bit symbols to provide corrections *(t)* of up to 10 errors per code block at data rates up to 320 Mbps. The encoder output code block will contain the unaltered original data symbols followed by the generated parity symbols. The decoder input contains the received data

and parity symbols including errors that may be introduced during transmission. Decoder output will be a completely corrected block or will be marked as non-correctable and the block will be output as received without any changes. Detailed pin-out and signal descriptions are described in [22].

4.2.3.2 Turbo Product Code CODECs

The Turbo Product Codes (TPC) are implemented using extended Hamming codes (or simple parity codes) in a two- dimensional or three-dimensional operation [17]. Encoding is performed by placing the data in an ($k \times k$) array, for a two dimensional code. Each row and column is then encoded with the appropriate extended Hamming code and the Error Correction Code (ECC) parity bits are appended to the end of each row. After all rows are encoded, the columns are encoded in the same manner resulting in a ($n \times n$) coded array.

The TPC algorithm applies an iterative decoding method to a product array of extended Hamming or single parity check codes. 'Turbo decoding' of a product code array involves individually decoding each row using a technique called soft decision correlation decoding [17]. The output of the row decoding is then combined with the original data and input to a decoder for each column using soft decision correlation decoding. The result of the column decoding is then input back to the row decoding. This process continues until the decoder settles on a valid transmitted code array or until the maximum number of iterations is reached. All of these operations are performed automatically within the TPC chip.

Figure 4.4 shows a block diagram of the TPC encoder and related modules. First, the CRC Engine computes and then inserts the CRC of the input data at the end of the each data block. The output of the CRC engine is then scrambled by exclusive-ORing it with the output of a pseudo-random binary sequence (PRBS) generator so as to ensure adequate bit transitions in the transmitted data stream. The scrambled data is then input to the TPC Encoder, which computes error correction code bits and inserts them at

appropriate locations in the data stream. Frame Sync insertion block inserts a programmable synchronization pattern into the bit stream. Finally, the Symbol-mapper formats the data stream to produce I and Q outputs for direct connection to the QPSK modulator.

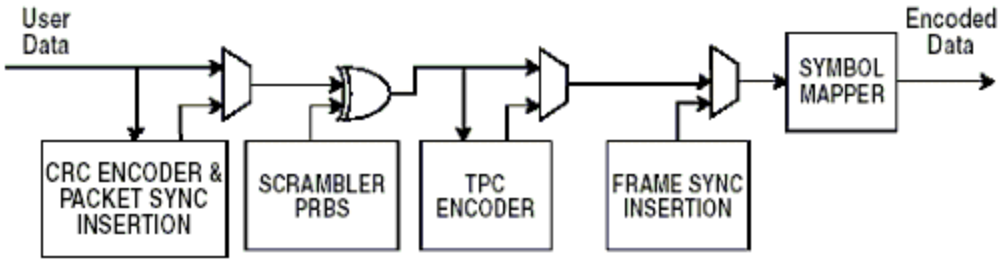


Figure 4.5 TPC Encoder Block Diagram

The TPC decoder path has a counterpart for every block on the TPC encoder path as shown in Figure 4.5. The channel interface formats the received channel data for decoding by the Turbo Product Code decoder. Since QPSK modulation is used, the soft (confidence) information comes directly from the in-phase (I) or quadrature (Q) component of the received symbol. The synchronization marks inserted at the transmitter end allows the TPC Decoder to determine the location of the first bit of the encoded block. After the TPC Decoder decodes the data stream, it is descrambled using a Pseudo Random Binary Sequence (PRBS) Descrambler. The CRC Engine then computes the CRC for each block and compares it to that appended to the data. The appropriate packet error signals are generated if there is a mismatch, and if no errors are detected, the decoded block is output.

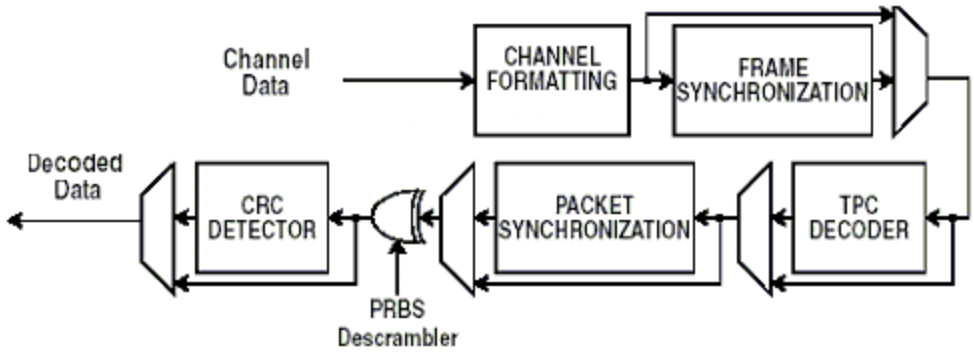


Figure 4.6 TPC Decoder Block Diagram

The TPC codec also has a microprocessor interface through which the processor can directly initialize it or change certain parameters such as block size, sync patterns, etc. The TPC Codec is programmed to implement a fixed (128,120)×(128,126) code.

4.2.4 External Dual-port SRAM

The external Dual-port SRAM allows simultaneous access by the processor and FPGA. The Dual-port SRAM can operate in either pipelined or flow through mode. In the pipelined mode, the read data access has a one-cycle latency while the flow through mode does not have the latency. However, the maximum frequency for the flow through mode is 50 MHz. Therefore, the FPGA port which reads/writes data at 40 MHz operates in the flow through mode whereas the processor port which reads/writes data at 66 MHz operates in the pipelined mode.

4.2.5 Network and I/O interfaces

The Gateway Controller is closely integrated with the other components of the gateway. The Gateway Controller uses standard well-defined network and I/O interfaces to communicate with the rest of the system consists of the following interfaces.

a) Modulator/Demodulator: The Gateway Controller provides control and power signals besides a QPSK-symbol mapped data interface. The modems connect to the Gateway Controller through custom back plane connectors [18].

b) Host Computer: The Gateway Controller communicates with the host/monitor Personal Computer (PC) using a customized ASCII Command set defined in [7]. The command set includes instructions to the Controller to change system parameters and to provide status information.

c) Fast Ethernet Interface: The gateway has a fast Ethernet wire-line interface and uses a RJ-45 connector.

d) Sounder Interface: The Sounder interface has not been fully defined for the initial stage of the controller specification. However, PECL outputs are available for use at a later stage. A possible Sounder interface is proposed in Appendix A.

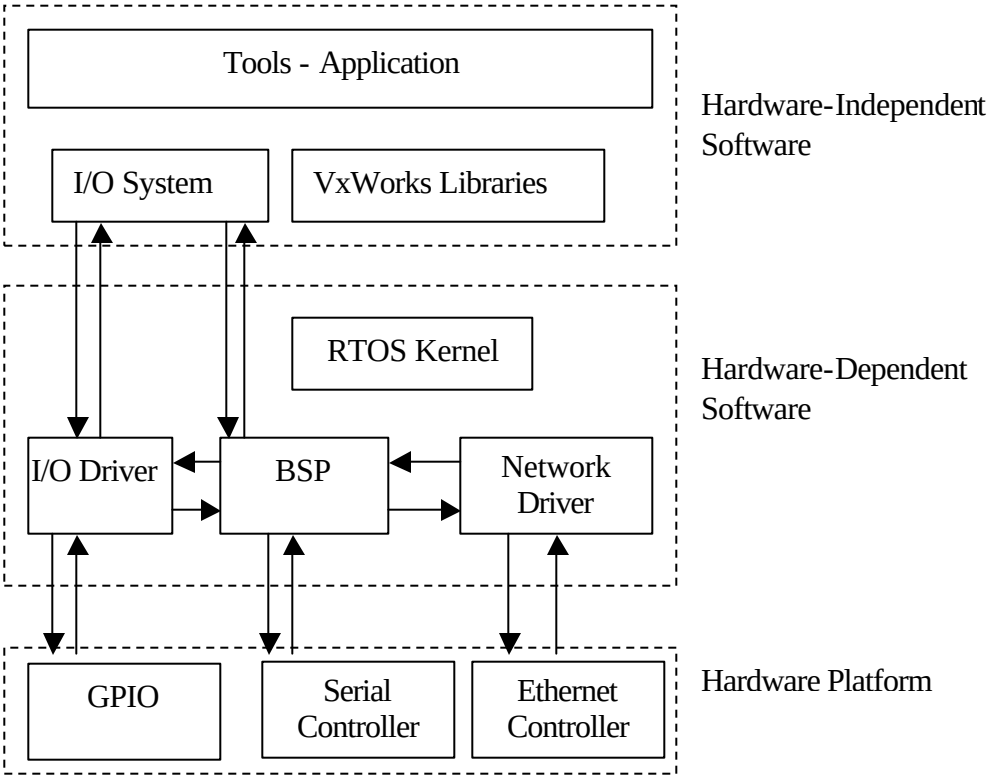


Figure 4.7 Software Platform Components

4.3 Software Platform

The software platform complements the hardware platform and consists of pre-integrated software tools, real-time operating system, and a design/development environment. The software platform is designed to be flexible and allow for rapid development and experimentation with different protocols and communication schemes. This thesis addresses a basic platform configuration to demonstrate the capabilities of the prototype hardware platform.

The components of a basic software platform and the interactions of the software platform with the Hardware platform is shown in Figure 4.7. The software platform consists of two types of components: Hardware Dependent and Hardware Independent. Some of the major components of the basic Software platform are given below and explained in detail in the following sections.

1. RTOS Kernel : VxWorks
2. I/O Drivers for peripherals: UART, Ethernet, etc.
3. Software Design Environment: Software Tools, Host IDE, MPC8260 ADS
4. In-Circuit Debug environment: JTAG/COP port, In-circuit emulator
5. Board Support Package

4.3.1 Real-time Operating System

The VxWorks RTOS is one of the most popular choices for embedded system designs. VxWorks [19] is a high-performance real-time operating system from Wind River Systems. The heart of the VxWorks RTOS consists of a multi-tasking kernel with interrupt-based, pre-emptive priority scheduling support, watchdog timers, and memory management.

4.3.2 Device Drivers

Device drivers are low-level software components that forge the actual connection between the microprocessor "engine" of the communications processor, and higher-level

software such as application tasks, communication protocol software and real-time operating systems. Many of these connections are unique to the target hardware and highly critical for system operation. The low-level software components that implement them are tedious to build because of the complexity and intricacy of the software-hardware interfaces specified by chip manufacturers. However, device driver templates that only require the software designer to modify the driver to suit the application are readily available for target systems such as the MPC8255 processor. This helps in reducing device driver development time by reuse of software elements designed by the chip manufacturer.

4.3.3 Software development tools

The development environment includes a full range of features from editors, compilers, simulators and source level debuggers to aid in efficient software development. The Tornado 2.0 [20] integrated design environment (IDE) from Wind River Systems has been chosen as the development environment. The Tornado IDE components execute on a host system with access to the Gateway controller target system. Application software modules written in C/C++ can be compiled with cross-compilers available in Tornado Host IDE for the MPC8255 CPU target systems. These application modules can take advantage of RTOS run-time libraries to reduce development times. An MPC8260ADS development board allows software developers to start software design before the hardware platform is completely developed.

4.3.4 In-circuit Debug Environment

The MPC8255 processor core has an internal common on-chip (COP) debug processor [13]. This processor allows access to internal scan chains through a JTAG/COP port for debugging purposes. It is also used as a serial connection to the core for emulator support. The JTAG/COP emulator running on the host system provides the developer with remote control and monitoring of target hardware to assist in board and system debugging.

4.3.5 Board support package

A board support package (BSP) is a collection of C and assembly routines that provide the RTOS with an interface to hardware. The VxWorks BSP [21] routines for the MPC8260ADS board are used as a template to create the BSP for the Gateway controller hardware. The BSP software is dependent on the hardware platform and directly interacts with the hardware.

4.4 Gateway Controller Application Software Modules

The Software platform provides a starting point for the development of the software. The Software platform allows for efficient reuse of code by providing pre-verified code for commonly used tasks. The application software development creates a unique instance of the Software platform. The Gateway Controller implements most MAC layer packet processing functions in software. This includes Ethernet packet processing and LMDS MAC packet processing. The Gateway Controller software modules are explained in detail in the following sections.

4.4.1 Transmit Process Software Modules

4.4.1.1 Algorithm for Ethernet Segmentation and LMDS MAC Data Payload formation

The Ethernet Segmentation process encapsulates the Ethernet packets into the LMDS MAC data payload for transmission over the LMDS wireless link.

1. Wait for IRQ from Ethernet PHY indicating the arrival of an Ethernet packet.

2. The Ethernet packet is moved from the Ethernet MAC buffer to the SDRAM memory. The newly arrived Ethernet packets enter at the end of an Ethernet packet FIFO in SDRAM memory from which the data payload is to be built. Update the size of the FIFO (ETHFIFO_Size) and number of Ethernet packets (ETHPKTNUM).

ETHFIFO_Size	
ETHPKTNUM	
PKT1_FRAG_OFFSET	
ETH PKT ADDR1	ETH PKT SIZE 1
ETH PKT ADDR2	ETH PKT SIZE 2
ETH PKT ADDR3	ETH PKT SIZE 3
.	.
.	.
.	.
ETH PKT ADDRn	ETH PKT SIZE n

Figure 4.9 Ethernet Packet Receive FIFO in SDRAM

3. Determine maximum data payload size (MAX_PAYLD) to fix the data payload buffer size in external dual-port SRAM memory where the data payload is to be built. Depending on the RS Coding level the maximum size of the data payload will vary. Use Table 4.1 for maximum payload size for each coding level. The size of the Ethernet packets can vary from 64 to 1518 bytes. The data payload is formed by filling the data payload buffer with multiple Ethernet packets and/or Ethernet packet fragments.

Coding Level	**RS Code RS (n, k)**	**Maximum Data (bytes)**	**RSMODE (0:2) register value**
1	RS (200,188)	1692	001
2	RS (200, 192)	1728	010
3	RS (200, 196)	1764	011
4	RS (200, 180)	1620	100
5	No Coding	1800	111

Table 4.1 Maximum size of data payload

4. In case the Ethernet packets are fragmented, an Ethernet Frame Fragment Header (EFFH) must precede each Ethernet fragment. To build the EFFH, the position of the first byte of the Ethernet fragment in the original Ethernet packet (EFFH_Offset) and the number of bytes in the fragment (EFFH_Size) must be calculated. The information must be included for the packet to be correctly re-assembled at the receiving end. If the fragment contains the last byte of the original Ethernet packet then the FIN bit of the EFFH (EFFH_FIN) is to be set.

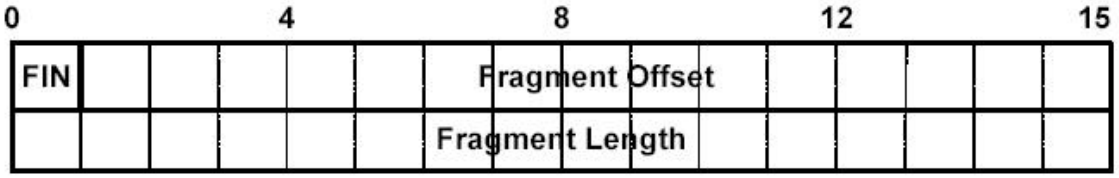


Figure 4.10 Ethernet Frame Fragment Header (EFFH) Format

5. If 200 bytes < ETHFIFO_Size < {MAX_PAYLD – (ETHPKTNUM * 4)}, i.e. all Ethernet packets and their Ethernet Frame Fragment Headers (EFFH) can fit in one data payload buffer of MAX_PAYLD size, then

(i) If ETHPKTNUM = 1, i.e. only one Ethernet packet is in the queue. Then insert one EFFH with EFF_FIN=1. Pad with (MAX_PAYLD – ETHFIFO_Size + 4) bytes of zeroes. Delete entry corresponding to the inserted packet from the FIFO. Update ETHFIFO_Size and ETHPKTNUM.

(ii) If ETHPKTNUM >1, then build EFFHs for each Ethernet Packet. Insert Ethernet packet/fragment and EFFHs alternately till the data payload buffer is full or all the packets/fragments have been inserted. The last EFFH must have EFF_FIN=1. Pad with {MAX_PAYLD – ETHFIFO_Size + (4* ETHPKTNUM)} bytes of zeroes. Delete entry corresponding to the inserted packets from the FIFO. Update ETHFIFO_Size and ETHPKTNUM.

6. If ETHFIFO_Size > MAX_PAYLD, i.e. all Ethernet packets in the FIFO will not fit within one data payload buffer. Identify the first 'k' packets of the Ethernet FIFO of size 'n' (i.e. ETHPKTNUM='n'), such that the sum of their sizes is as close to [MAX_PAYLD –(k*4)] as possible without exceeding it.

(i) If Σ ETH PKT SIZE(i=1 to k) = MAX_PAYLD – (k * 4). The first k packets and corresponding EFFHs are inserted and no padding is necessary. Set EFF_FIN=1 for the k^{th} EFFH. PKT1_FRAG_OFFSET should be cleared.

(ii) If Σ ETH PKT SIZE(i=1 to k) < {MAX_PAYLD – [(k+1) * 4]} < Σ ETH PKT SIZE (i=1 to k+1). Same as Step 6a) but now pad the payload buffer with a fragment the $(k+1)^{th}$ Ethernet packet and its EFFH. Update ETHFIFO Size and ETHPKTNUM. Set top of FIFO to $(k+1)^{th}$ packet and PKT1_FRAG_OFFSET to indicate the position from which the next payload must begin reading the first Ethernet packet in the queue.

7. If ETHFIFO_Size < 200 bytes then

(i) If PKT1_FRAG_OFFSET is set then, insert the fragment of the first packet and all remaining packets into the data payload buffer and pad the rest of the buffer with zeroes.

(ii) If PKT1_FRAG_OFFSET is cleared then Goto Step 1

8. The data payload of size MAX_PAYLD is now completely formed and resides in the data payload buffer in external dual-port SRAM. The Beginning Address of the buffer must be written to the DMA1 register (DMA1_BEGADDR). Set RSMODE Register values based on the coding level according to Table 4.1.

9. Inform the Transmit Control Process that the data payload is ready.

4.4.1.2 Algorithm for LMDS MAC Header formation

The LMDS MAC Header consists of several fields as shown in Figure 4.10. To build the header the information related to these fields, listed in Table 4.2, must first be collected from or updated by various sources. The fields relating to acknowledgements need to be updated by the Receive process. The transmission time slots on the uplink are statically distributed among 'n' remotes during initialization and remain fixed. The hub is allocated all time slots on the downlink. A Data frame consists of 'n' time slots one for each remote.

Information	**Source**
Acknowledgement Service ON/OFF?	System Parameters from Monitor PC
Time Slot number	System Parameters from Monitor PC
FEC Level	System Parameters from Monitor PC/ LMDS Data Payload formation process
Sequence Nos. for Acknowledgement	Timing and Control Process
Acknowledgement Updates	Receive Process

Table 4.2 Information Required by LMDS MAC Header Formation Process

1. Set the Time Slot (TM_SLOT) field to indicate the time slot at which the packet is transmitted.

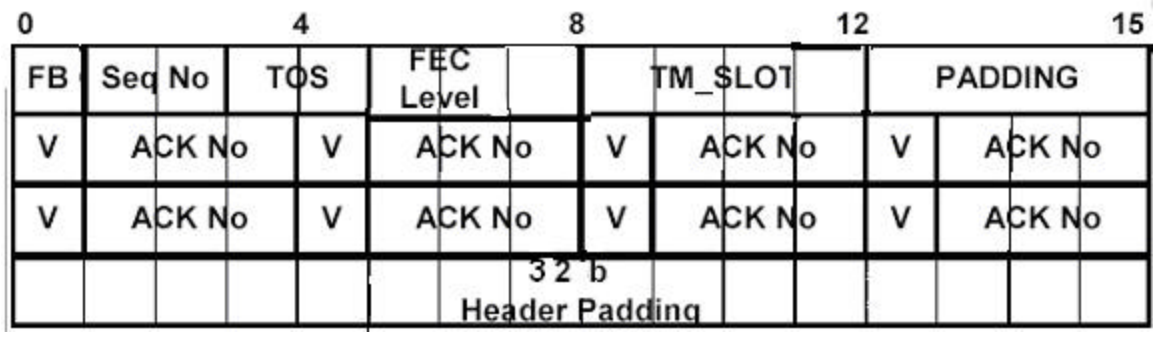


Figure 4.11 LMDS MAC Header

2. For a normal time slot, set the Frame Border (FB) field shown in Fig 10.3 to $(0)_{2.}$

3. Sequence Number (SEQ NO) is required for Acknowledgement Service. If the Acknowledgement Service is turned on, then the field can contain a value between 0 and 3. If the Acknowledgement service is turned off, then the field must be filled with zeroes.

4. The Type of Service (TOS) indicates if the Acknowledgement service is to be turned on or off. $(00)_2$ indicates Acknowledged Service and $(11)_2$ indicates Unacknowledged Service. Values of $(01)_2$ and $(10)_2$ are invalid.

5. The FEC Level field contains the coding level of the data payload (same as RSODE register values). This is required to correctly program the RS Decoder at the receiver to decode the data payload. For the valid RSMODE register values refer Table 4.1.

6. Acknowledgement Number (ACK NO) and Valid (V) fields are used to piggyback acknowledgements of LMDS MAC packets received by the receive process. The ACK NO field is the sequence number of the next expected transmission slot. There are eight optional ACK NO and V fields. The ACKNO fields are valid only if the V fields are set to 1.

7. Since the FEC Level field is to be updated based on the coding to be used on the data payload, the header formation should be performed only after data payload formation parameters are fixed or preferably after the payload formation process. This also allows more time for the Acknowledgement updates from the receive process to be added.

8. The header field bits from 13 to15 are reserved for future use and must be padded with zeroes. The RS Encoder requires a minimum of 10 bytes for Encoding. So the header is padded with 4 bytes of zeroes to make the header size equal to 10 bytes.

9. The completely formed MAC header is stored in the external dual-port SRAM memory. The beginning address of the MAC header is programmed into the

DMA1_BEGADDR register and RSMODE register value is set at “000” for the Header RS Encoding. The Header is always encoded at a fixed coding level using RS(10,18) code.

4.4.1.3 LMDS PHY Transmission Scheduler

Although the hardware modules implemented in the FPGA can operate independently without supervision from the processor, the processor keeps close tabs on the PHY layer processing. The processor initiates every stage of the PHY processing using a DMA_START signal. At the end of the processing stage the hardware processing stage issues an IRQ (DMA_END) to the processor. The hardware modules also notify the processor if any errors were generated during the processing stage.

1. Wait until payload data is available at the dual-port SRAM.

2. Set M1_BEGADDR with the beginning address location of the data payload buffer in external dual-port SRAM (DPSRAM). Set M2_BEGADDR Register with the beginning address of the internal DPSRAM (M2) where the RS Encoded Data is to be stored. Set RSMODE register based on FEC Coding level to be used.

3. Assert DMA1_START GPIO signal. For a description of the DMA1 process refer to Section C.3.1.1. Wait until the DMA1_DONE (DMADONE_IRQ1) is generated.

4. Check if any errors were generated by reading DMA1 error register (DMA1_ERROR). If errors were present, fix error source and repeat Step 2. If no errors were received proceed to Step 5.

5. Set M1_BEGADDR with the beginning address location of the MAC header buffer. Set M2_BEGADDR so that the header and payload are stored in contiguous locations in the internal DPSRAM. Set RSMODE register to $(000)_2$.

6. Assert DMA1_START GPIO signal. Wait until the DMA1_DONE (DMADONE_IRQ1) is generated.

7. Check if any errors were generated by reading DMA1 error register (DMA1_ERROR). If errors were present, fix error source and repeat Step 5. If no errors were received proceed to Step 6.

8. Verify if the Turbo Product Code (TPC) Encoder has been initialized. Assert DMA2_START GPIO. For a description of the DMA2 process refer to C.3.1.2.

9. Wait until the Transmission Timer indicates the beginning of the transmit slot.

10. Assert TXPRE_START signal of the Preamble Generator process to start transmission of the preamble. The preamble is a 1000 symbol sequence with alternate 0s and 1s. This is necessary to guarantee that the Demodulators at the receiver are synchronized with the Modulators at the transmitter. Refer to Section C.3.1.3 for a description of the Modulator interface process (MODIF).

11. Wait for preamble transmission to complete (16.66 μs), and then assert TXDAT_START to signal beginning of LMDS PHY frame.

12. Wait for TX_DONE interrupt request (IRQ) or TX_ERR IRQ signals. TX_DONE is asserted then the data was transmitted successfully. Schedule next payload for transmission.

13. If the TX_ERR signal is asserted then the data payload must be scheduled for retransmission.

4.4.2 Receive Process Modules

4.4.2.1 Algorithm for LMDS PHY Receive Control Process

The Receive control process performs functions that are similar to that of the Transmission Scheduler. The receive control process keeps track of the receiver hardware processing stages.

1. Initialize the TPC Decoder registers. Activate the Demodulators and other hardware receiver modules by asserting the LISTEN signal. For a description of the Demodulator interface (DEMODIF) process refer to Section C.3.2.1.

2. Program the M3_BEGADDR Register with the starting address location where the received packet is to be stored in internal DPSRAM (M3).

3. Wait until TPC Decoder issues an IRQ to signal that data is available at the decoder outputs. Check TCDECDONE_ERR and TCDECDONE_NOERR IRQs. If TCDECDONE_ERR packet is asserted then the packet is lost.

4. If TCDECDONE_NOERR is asserted, check the TPC Decoder Error Register for uncorrectable errors in the received LMDS PHY packet.

5. Assert DMA3_START GPIO to initiate the RS Decoding of the MAC header by DMA3.

6. Notify Header Decoding process of arrival of new packet header.

7. Wait until header is decoded and the Header Decoding process provides information on the payload FEC level.

Coding Level	RS Code RS (n, k)	DMA3_DATMODE(0:2) register value
1	RS (200,188)	001
2	RS (200, 192)	010
3	RS (200, 196)	011
4	RS (200, 180)	100
5	No Coding	111

Table 4.3 Maximum size of data payload

8. If Step 4 indicated uncorrectable errors and the RS coding level is 5 (no RS coding), then discard packet. Notify Ethernet Re-assembly process.

9. If Step 4 indicated uncorrectable errors but RS coding levels are between 1 and 4, then program DMA3_DATMODE registers to indicate FEC level. The values for DMA3_DATMODE registers are shown in Table 4.3

10. Assert DATA_START GPIO to initiate payload RS Decoding. See section C.3.2.3 for details on the DMA3 process.

11. Wait until DMADONE_IRQ3 (DMA3_DONE) is asserted.

12. Program M1_DMA3_BEGADDR Register with the starting address of the receive data payload buffer. Assert DMA3O_START GPIO. See Section C.3.2.4 for details on the DMA3 process.

13. Wait until DMA3O_DONE is asserted. Read RSDEC_STATUS Registers.

14. If the payload contains uncorrectable errors after RS Decoding, discard packet. Notify the Payload decoding process.

15. If all errors were corrected after RS Decoding then, notify the payload decoding process to start after providing the starting address of the data payload buffer.

4.4.2.2 Algorithm for LMDS MAC Header Decoding Process

1. Wait until Receive Control Process indicates that a new packet has arrived.

2. Read first bit of the payload. For a normal payload header the bit should be 0. Go to Step 4.

3. If first bit is 1 then, stop processing. Notify Frame Control Header Decoding process. Go to Step 1.

4. Read first byte of the payload header. Forward bits 5-8 (FEC level field) of the first byte to the Receive control process.

5. Read all other fields. If Acknowledge service is specified in TOS field, notify Header Formation process and provide it with SEQ NO, TM_SLOT information.

6. Check V field values. If V field value is 1 then read the corresponding ACK NO field. Send updates on the acknowledgements to the Transmission Scheduling process.

4.4.2.3 Algorithm for Ethernet Re-assembly and Payload Decoding process

The Ethernet Re-assembly process reads the payload stored in the external DP SRAM memory into Ethernet packet transmit buffers in the SDRAM memory. A new Ethernet packet transmit buffer is created for every Ethernet packet. Once the Ethernet packet is completely reassembled then the packet is moved to the Ethernet packet transmit FIFO and schedule for transmission by the Ethernet MAC driver.

1. Wait until the Receive Control process indicates that a payload has arrived in the data payload receive buffer. Create an Ethernet packet buffer and an Ethernet packet transmit FIFO.

2. Read the EFFH field.

(i) If Fragment Offset (EFFH_Offset) field is 0, create a new Ethernet packet buffer, Update the Ethernet transmit FIFO.

(ii) If EFFH_Offset is non-zero then continue to store in the same Ethernet buffer.

(iii) Read the next EFFH_Size bytes of data and store it in the Ethernet buffer with appropriate offset (EFFH_Offset).

(iv) If EFFH_FIN is set to 1 stop processing payload. Go to Step 3.

(v) If EFFH_FIN is 0 then read next EFFH. Repeat Step 2.

(vi) If Ethernet Packet transmit FIFO contains an Ethernet packet then notify Ethernet driver to schedule a transmission.

4.5 Summary

The Hardware platform is designed using configurable hardware components to allow maximum flexibility and to help exploration of the protocols. The tradeoffs in implementing the processing modules in hardware or software are presented. The Software platform simplifies the application development by providing tools and software code libraries. The software developer can reuse the hardware independent components of the Software platform and only needs to port the hardware dependent components of the system for new applications or designs. The porting of the hardware dependent platforms such as the BSP is also simplified as the developer already has a basic framework from which he can work and does not have to start from scratch.

CHAPTER 5

Hardware and Software Testing

5.1 Hardware Design Verification

Overall hardware implementation of the controller design consists of the entry of the conceptual design into electronic description format (design entry), conversion of the design into a logic level form (synthesis), and translation of the design into the physical FPGA specific component placement and signal routing (implementation). The design verification process consists of testing the design for conformity at several intermediate stages. The verification steps performed after each major stage of the design are shown in Figure 5.1 and include: behavioral or functional simulations, synthesis checks, post-synthesis timing verification, and post-implementation timing verification. All of these steps are done using simulation tools like Synopsys' VHDL Compiler and tool suite [23] and synthesis tools like Synplicity's Synplinfy [24] and Xilinx's Foundation ISE Tools [25].

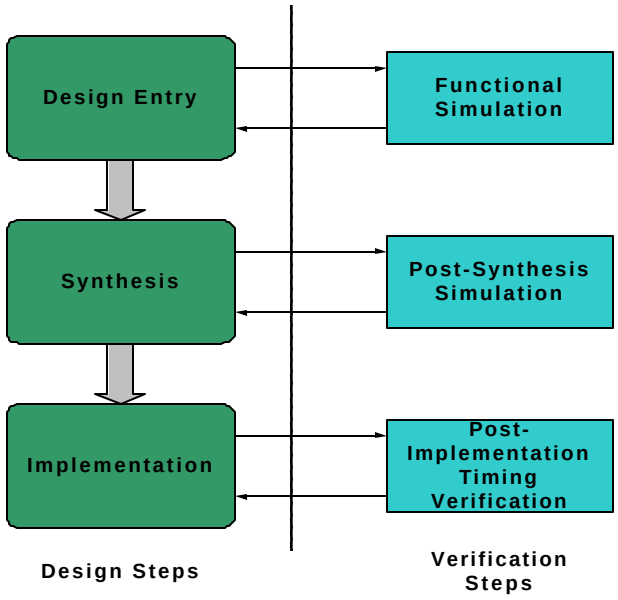


Figure 5.1 Hardware Verification steps after each design stage

5.1.1 VHDL Behavioral Description

The FPGA logic is designed employing behavioral VHDL that can be used for behavioral simulation as well as FPGA synthesis. The VHDL descriptions are built based on finite state machine (FSM) descriptions of the logic modules. The designs are based on a hierarchical structure. The FSM descriptions are provided in Appendix B. A test bench "wrapper" that abstracts some of the higher-level processor-based software interactions is used to functionally simulate the top-level VHDL module. This test bench provides the simulated status and control stimuli that are provided by the processor in the physical implementation of the controller.

5.1.2 VHDL Behavioral Simulation

The first step in the FPGA hardware verification process is to devise high-level language-based verification models and methodologies. A VHDL-based behavioral model for the system architecture was developed with high-level system behavioral modules, including: "Bus-Functional" processor modules, DRAMs, SRAMs, etc. The VHDL behavioral model helps to create exhaustive HDL-based methodologies to verify bus and chip-level specifications. A rudimentary Bus Functional Model for the MPC8255 60x bus and Local Bus is used in the system model to verify the bus interface functions in the FPGA and peripheral chips.

5.1.2.1 Simulation of Embedded memory Interactions

The simulation system model consists of synchronous memory in the form of dual-port SRAMs. The Virtex FPGAs have built-in embedded memory that can be tailored to the required configuration. The embedded memory can be synthesized on the FPGA using programmable, but pre-configured, IP cores in the form of EDIF files. The CORE Generator tool from the Xilinx 2.1i Alliance Series software suite was used to design and generate the memory modules. By default, RAM cores have all their contents initialized to zero by the CORE Generator. However, sometimes the RAM module is

required to have specific data stored in it for simulation. There are two ways of specifying memory contents with the CORE Generator.

- Memory Initialization File (.MIF) file – The .MIF file is an ASCII file in which each line of text specifies the content of a RAM location. The memory contents are specified as binary digits with one line of text corresponding to every address location in memory. By default the Xilinx Core Generator creates a .MIF file to specify the RAM contents and initializes all locations with zeroes.

- Coefficient (.COE) File - Allows the RAM locations to be specified in hexadecimal format. The .COE format is closer to Intel hex format used for ROMs. The .COE format was not used for the simulations.

For purposes of simulation, a behavioral model of the embedded dual-port SRAM is selected from a library of Xilinx Virtex primitives. The primitives are VHDL behavioral models that mimic the behavior of the gates in the EDIF net list of the IP core. The SRAM model can then be instantiated in the simulation system model just as any other module of the system.

Shown in Figures 5.2 and 5.3 are the beginning and end, respectively, of a memory read transaction by DMA1. The dual-port RAM Memory (M1) being accessed is synchronous, i.e. all control signals and data signals are valid at the rising edge of the clock. The clock signal is added to the simulation waveform to show the beginning instant of each clock cycle. The SRAM is four bytes or 32 bits wide. But the DMA1 accesses only a byte or eight bits at a time as it directly transfers the read byte to the 8-bit input bus of the RS Encoder. To read one byte per clock cycle the Byte Enable (M1L_nBE[3:0]) control signal is used. The data bus will contain only the bytes that were selected by de-asserting the corresponding Byte Enable bit, i.e. a value of $(0111)_2$ or $(7)_{16}$ will select only the first byte, a value of $(1011)_2$ or $(E)_{16}$ will select the second byte and so on.

The BYTCOUNT and PARCNT_TEMP counters keep track of the number of cycles for which data is to be read from memory. In this simulation a RS (200,186) code was used, hence a block of 186 data bytes are to be read from memory. This process is repeated nine times till eight blocks of data are input to the RS Encoder. The memory data bus shows $(FFFF)_{16}$ since the memory model is initialized to all ones. Note that the address is loaded at the beginning of the DMA1 process (at ~310 ns in Figure 5.2) by de-asserting M1L_nADS. After this the internal address counter in the memory is used. The address counter is incremented by de-asserting M1L_nCNTEN after every four read cycles.

By using these control signals all four data bytes stored in every location is read. The simulation waveforms show that the appropriate memory control signals are generated accurately and that the memory read cycles are in turn performed correctly.

5.1.2.2 Simulation of External Interfaces

Several processes on the test bench wrapper VHDL module are used to represent each of the non-FPGA hardware modules or interfaces. The test bench is designed to behaviorally respond for the external modules such as the Reed Solomon Codec and TPC Codec. For example, though the Reed-Solomon encoder process module does not actually perform Reed Solomon encoding, it generates the appropriate control signal responses and correct number of random data outputs based on the input control signals of the physical Reed Solomon CODEC chip. The test bench provides an abstract interface to the logic external to the FPGA, which is adequate to verify the functionality of the FPGA modules and interfaces.

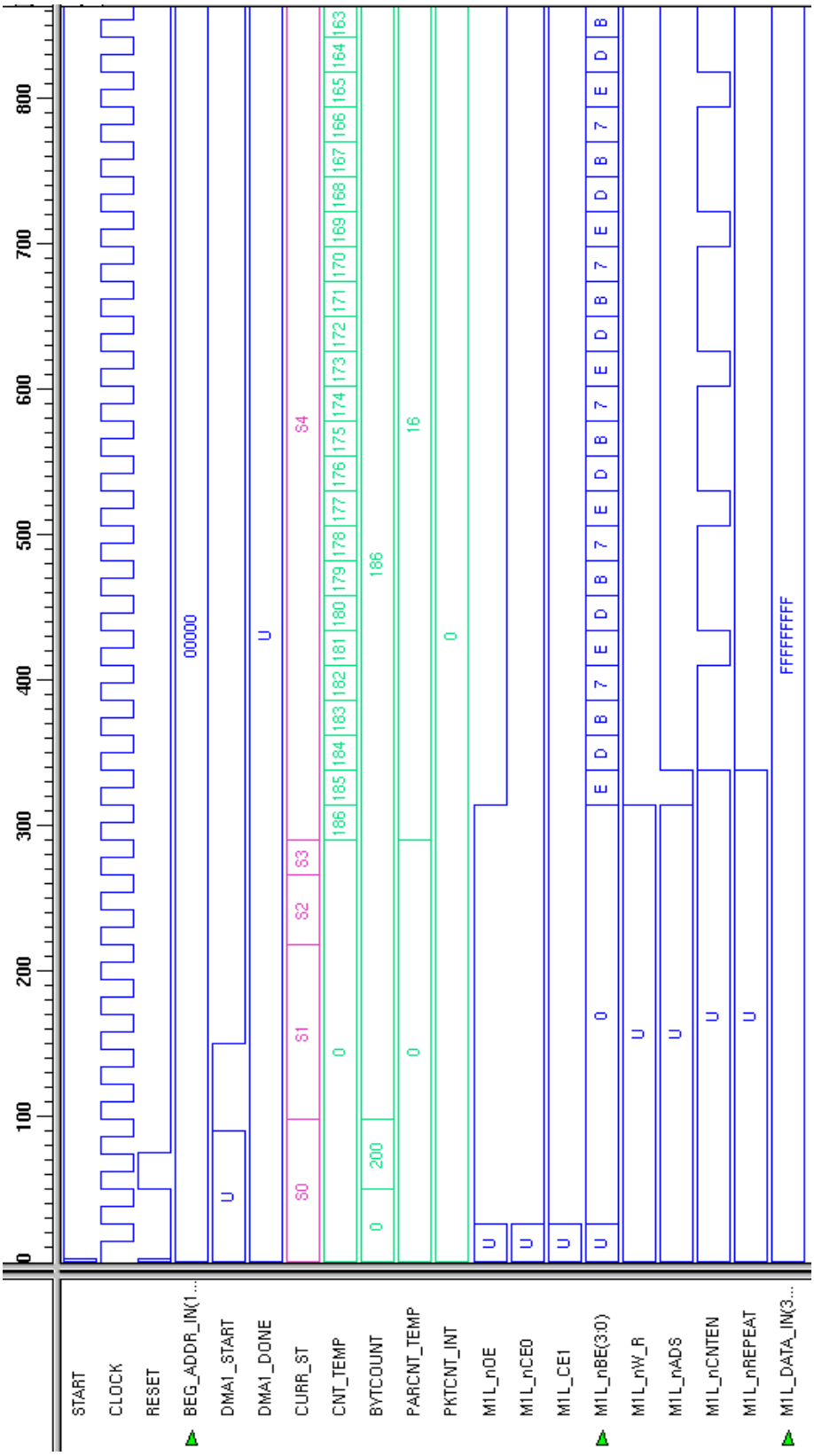


Figure 5.2 Simulation waveform view of "Memory Read" with memory models (1 of 2)

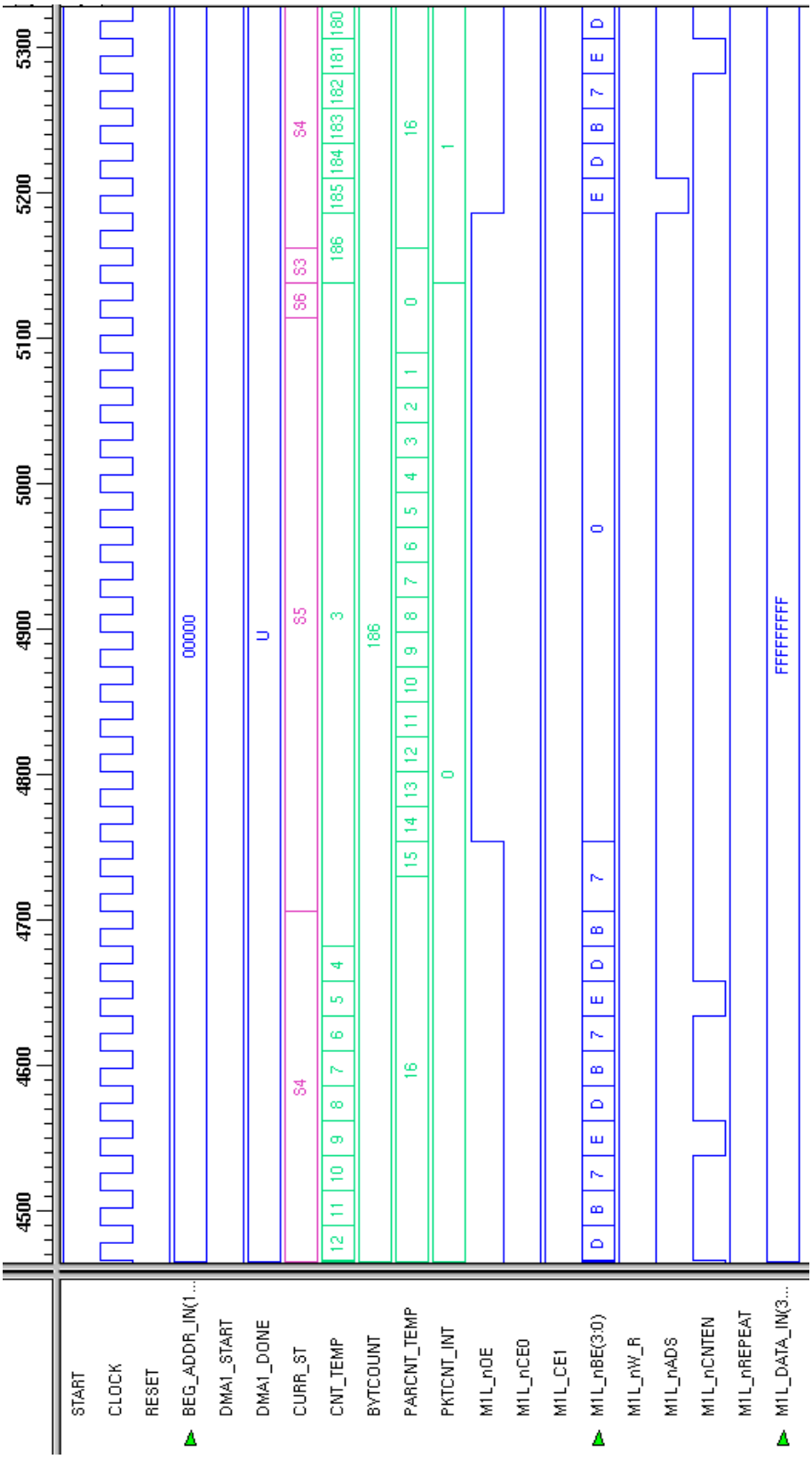


Figure 5.3 Simulation waveform view of "Memory Read" with memory models (2 of 2)

Shown in Figs. 5.4 is an example of an interaction of FPGA DMA with the Reed Solomon Encoder that is implemented as an external ASIC. The DMA provides the RS Encoder with latency configuration information using RSENC_TA[3:0]. The RSENC_RESET signal resets all the RS Encoder configuration and internal registers. The RSENC_CLK is a 40 MHz clock input to the RS Encoder. All signals are read or written at the rising edge of RSENC_CLK. The RS Encoder must first be initialized before the Encoding process can begin. The initialization process consists of two steps and begins immediately after the processor asserts the DMA1_START signal. The first step consists of de-asserting RSENC_ENIN and RSENC_RESET for four clock cycles. The second step consists of asserting RSENC_RESET for two clock cycles. The RSINITCOUNT counter is used to keep track of the initialization steps.

When the initialization process is complete, the RSENC_ENIN and RSENC_ENOUT signals are asserted simultaneously indicating to the RS Encoder that the first byte of data is available at the input. The RSENC_DIN[7:0] is the data input bus to feed data bytes from memory for RS encoding. The memory locations are filled with all ones. Hence the inputs to the RS Encoder is always $(11111111)_2$ or $(FF)_{16}$. The RS Encoder process as mentioned earlier does not perform actual RS Encoding. It merely generates the handshake signals and inverts every alternate bit input to it and the RSENC_DOUT[7:0] contains $(01010101)_2$ or $(55)_{16}$. This is used to distinguish the input and output to the dummy RS Encoder process and verify that it acted upon the data. The RS Encoder process correctly introduces a latency of three clock cycles before asserting RSENC_RDY and making the data output available on RSENC_DOUT. Since the simulation aims to verify the DMA interface functionality rather than the functionality of the RS Encoder this arrangement is found to be sufficient.

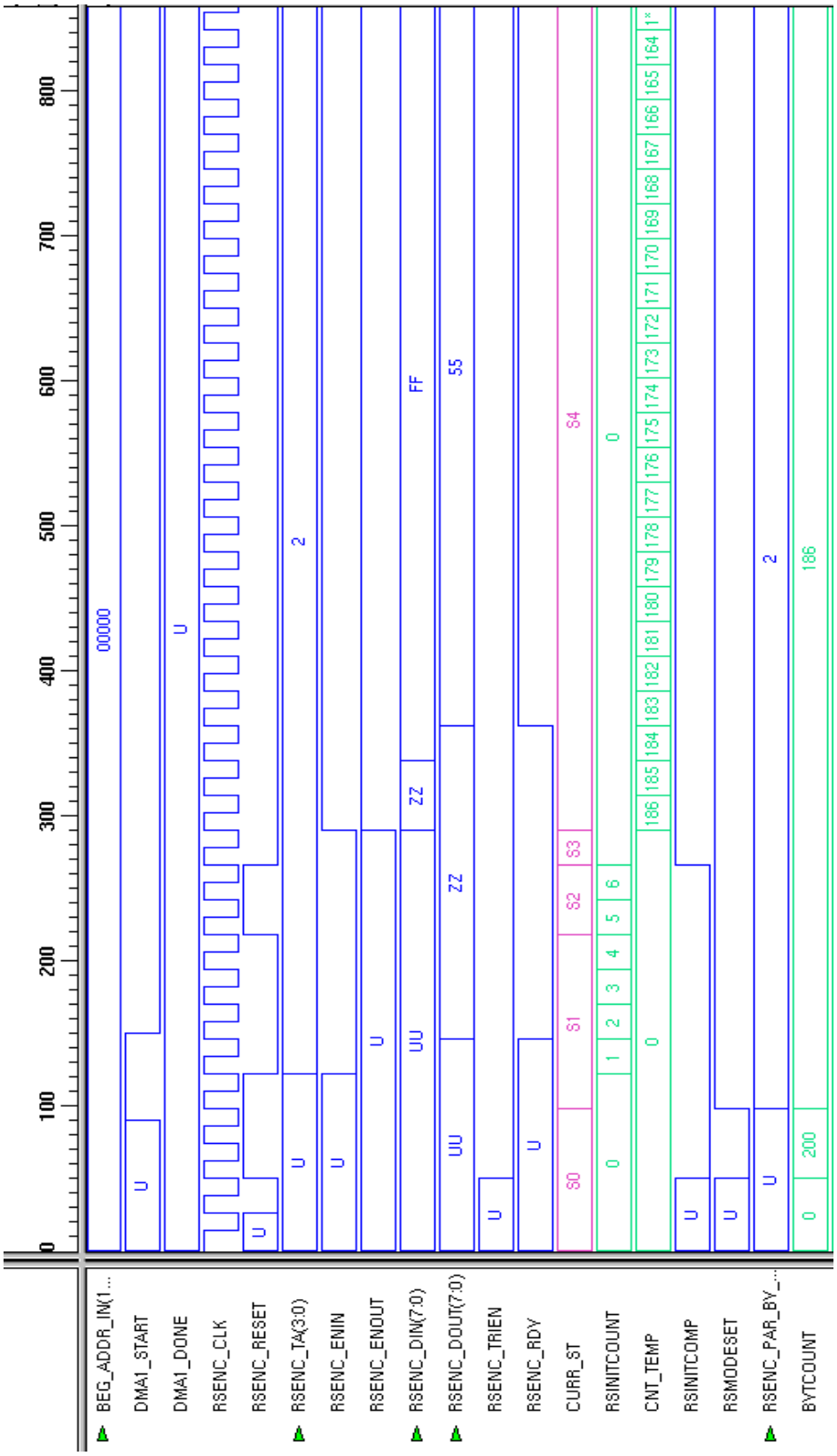


Figure 5.4 Simulation waveform view of external FPGA interfaces

5.1.2.3 Simulation of Processor Interactions

The processor initiates a DMA process with a DMA Start signal and then keeps track of the progress of each DMA packet processing stage. Each DMA in turn notifies the processor upon completion of its task with a DMA Done signal. The processor then requests the next DMA processing stage to start its operations. The notification to the processor is in the form of a processor Interrupt Request (IRQ). The processor notifies DMAs through its general-purpose I/O (GPIO) ports and through registers on its local bus. To simulate the entire transmit and receive paths as one continuous process the processor general-purpose inputs and outputs stimulus must be generated by some other means. A VHDL functional block that mimics the behavior of the relevant processor module, the processor GPIO controller in this case, is created in the test bench wrapper.

The test bench process is used to provide the general-purpose I/O (GPIO) signals that would be generated by the I/O control software running on the processor. An example of a DMA Start and a DMA Done signal are the DMA1_START and DMA1_DONE signals respectively. Both signals belong to the DMA1 process and are shown shown underlined in red in the waveform in Figure 5.5. The test bench process asserts the DMA1_START signal to initiate the DMA1 process and then de-asserts it. The assertion of the DMA1_START is shown in Figure 5.4 from the 90ns to 150 ns marker.

On receiving the DMA1_START signal the DMA1 begins to input data to the RS Encoder. The DMA1 must transfer 9 blocks of data for encoding and maintains a counter (RSPKTCNT _INT). After all the transfers are completed successfully the DMA1 process generates a DMA1_DONE signal that is connected to a processor IRQ. The generation of the processor IRQ is marked by the circled portion of the waveform in Figure 5.5. Normally the IRQ would require some processing time in the software. But to keep simulations low, the test bench process issues the next DMA start signal immediately.

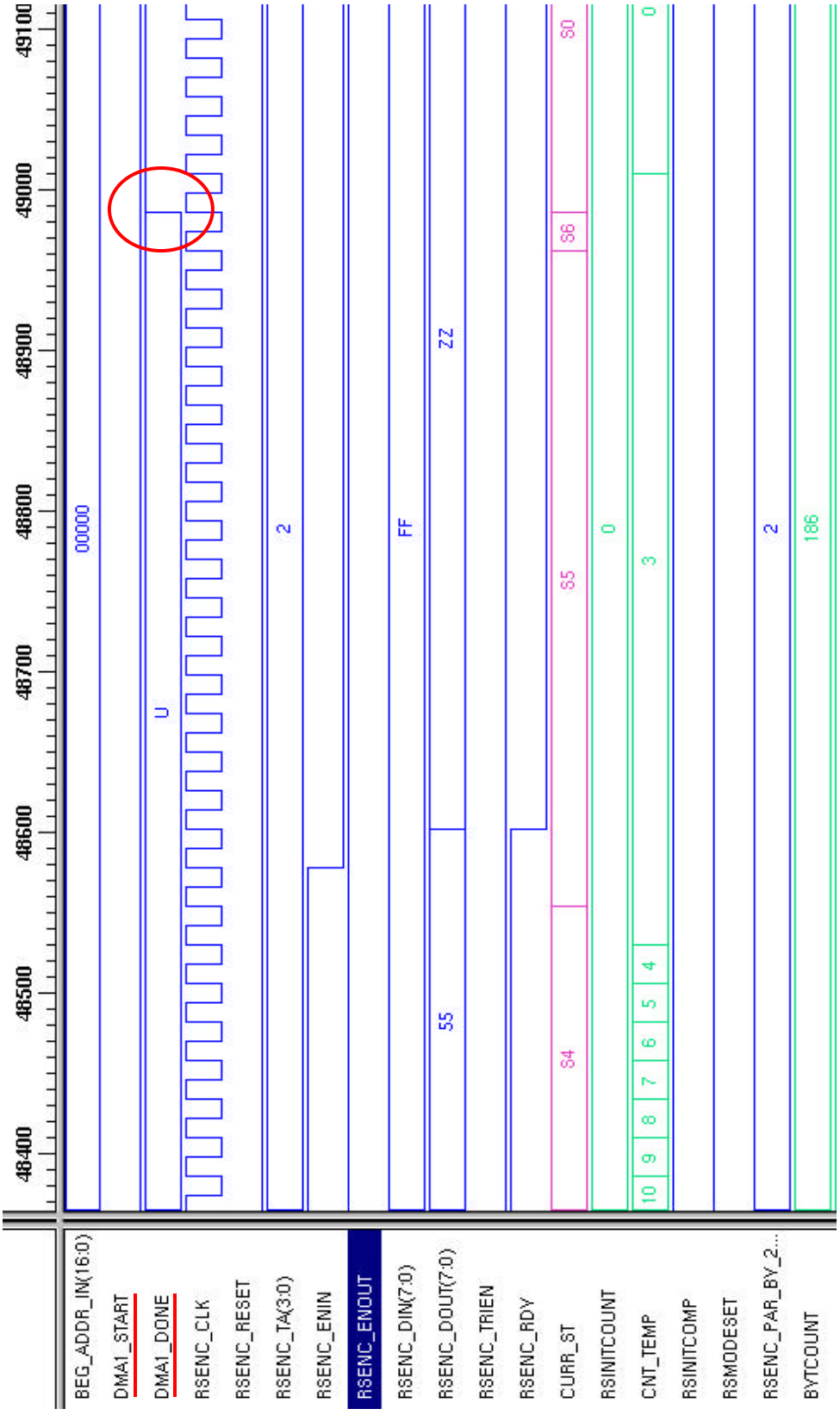


Figure 5.5 Simulation Waveform View of DMA processor interface and IRQs

5.1.3 FPGA synthesis environment

Synplify v7.0 [24] synthesis tool from Synplicity and ISE Foundation Tools v4.2i [25] from Xilinx were used for synthesis. The synthesis constraints for the design modules were setup in constraint definition files (.SDF). The physical part targeted was the XCV600FG680. Fig.5-7 shows the Synplify synthesis tool interface that reads in VHDL source files and Electronic Data Interchange Format (EDIF) files and then successfully placed and routed the FPGA VHDL-specified modules, IP cores, and the memory modules into a single EDIF netlist. The EDIF file format provides a common format for porting the design across multiple synthesis tools. The EDIF file for the FPGA co-processor design (Modem_top.edf), produced by the Synplify Tool after completing the synthesis process, is used as an input to the Xilinx ISE Foundation tools.

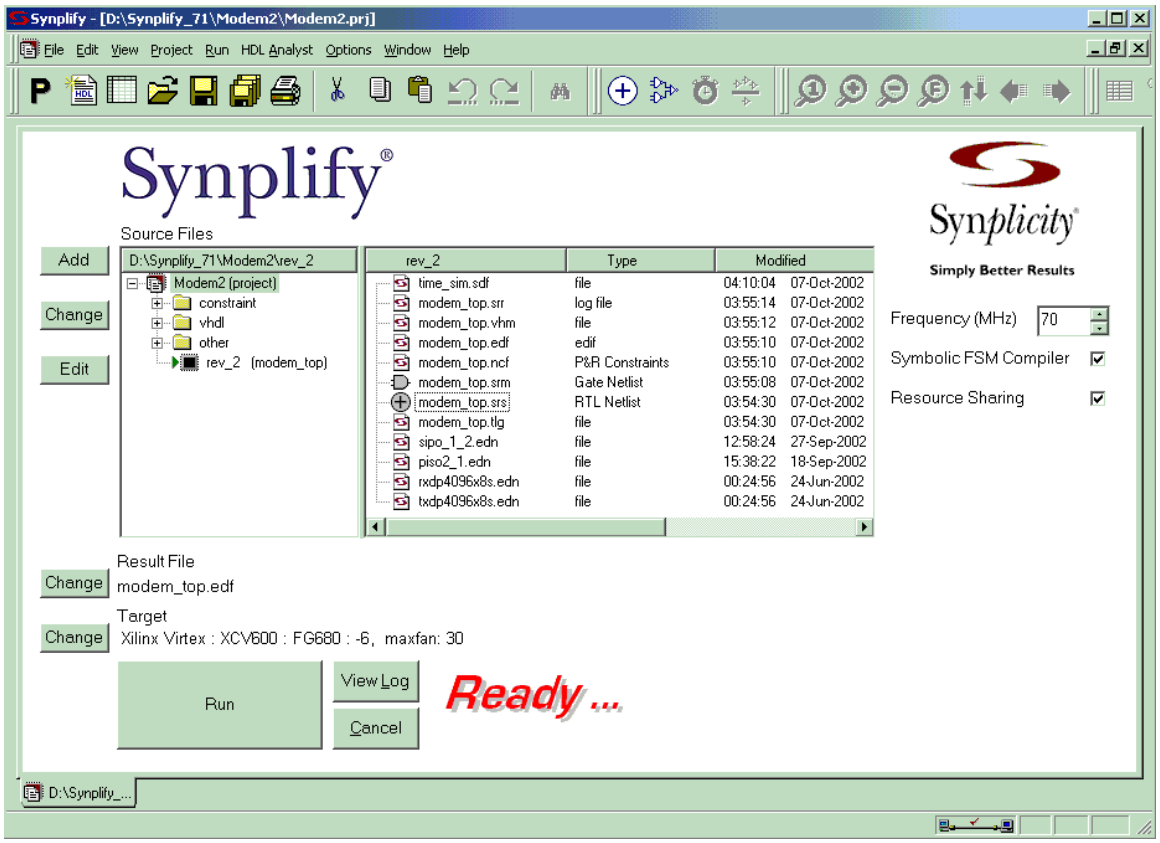


Figure 5.6 Synplify synthesis tool

The Xilinx ISE Foundation tool suite contains many utilities and programs that are used for mapping, place and route (P&R), timing simulation etc. The tools and their outputs are explained below. The Xilinx Design Manager utility shown in Figure 5.8 is used to maintain version information and to track changes for each iteration of the synthesis process. It also provides a Report Browser interface that organizes the various output report files in an easily readable format.

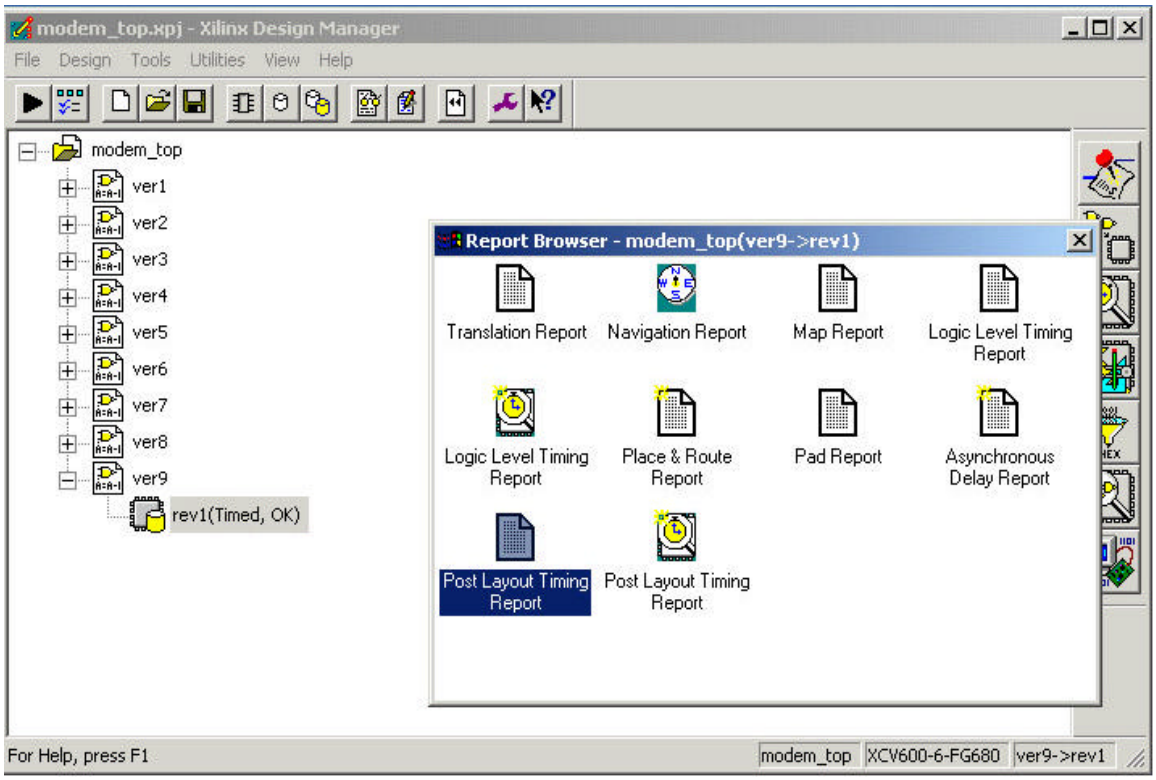


Figure 5.7 Xilinx ISE Tool performs P&R and generates several reports

The FPGA device utilization figures we achieved are detailed in the output file from the "Mapper" program. A summary of the map report is tabulated in Table 5.1. The Mapper report calculates the propagation delays of the mapped design and reports any violations of the setup and hold time constraints. The report file indicated no errors in the design.

The map report also contains device information and design summary details. The device information specifies the target devices characteristics for which the design mapping process was performed. In this case a XCV600 FG680 with a speed grade of –6 was used. The speed grade of the FPGA is specified by the manufacturer and is measure of FPGA gate delay times. The high-speed grade was chosen in consideration of the use of high-speed clocks in the system.

The device utilization summary shown in Table 5.1 is useful to determine if the FPGA resources are used efficiently. The total equivalent gate count for the design and IOBs are close to 290K gates.

The LUT and Slice utilization factors are low at 5% and 8% respectively. It was decided not to scale down to a smaller FPGA based on the application considerations. The Gateway Controller is designed to allow flexibility in exploration of different wireless MAC protocols. Later versions of the protocol may require additional IP cores to be implemented in the design. Another advantage of the low utilization factors is the low usage of interconnect resources. This affords a lot of flexibility in redesigning the FPGA logic without causing costly board redesigns. Also note that the IOB utilization is currently 54%. This allows for expansion of newer signals to be added to later design revisions by activating the unused pins of the FPGA connected to the unused processor GPIO.

Since the architecture is designed to be memory-centric, most of the internal BlockSRAM resources were utilized resulting in a high utilization factor of 66%. Most of these resources were used as internal dual-port SRAMs along the transmit and receive paths. Also all the clock buffers or CLKIOBs are utilized in the design. This is because of the use of several clocks in the design. In fact some of the slower clocks had to be distributed by ordinary IOBs so that the high speed CLKIOBs can be reserved for distributing high-speed clocks.

Design Information		
Target Device	XV600	
Target Package	FG680	
Target Speed	-6	
Mapper Version	Virtex Revision 1.58	
Number of Errors	0	
Design Summary		
Resource	**Utilization**	**Utilization %**
Number of Slices	620 out of 6,912	8%
Number of Slices containing unrelated logic	0 out of 620	0%
Number of Slice Flip Flops	640 out of 13,824	4%
Total Number 4 input LUTs	819 out of 13,824	5%
Number used as LUTs	782	
Number used as a route-thru	37	
Number of bonded IOBs	278 out of 512	54%
IOB Flip Flops	257	
Number of Tbufs	291 out of 7,104	4%
Number of Block RAMs	16 out of 24	66%
Number of GCLKs	4 out of 4	100%
Number of GCLKIOBs	4 out of 4	100%
Total equivalent gate count for design	276,157	
Additional JTAG gate count for IOBs	13,536	

Table 5.1 Xilinx Mapping Report File summary for design

5.1.4 Post-synthesis Timing Simulation

The Xilinx Foundation utility "ngdanno" produces the standard delay format (.SDF) file, which must be back-annotated with the FPGA netlist for gate-level simulation. The "ngd2vhdl" program produces a VHDL netlist of the SimPrims primitives for vhdl gate-level simulation. In addition to producing an EDIF file of the synthesized wrapper for place and route, synthesis also produced a vendor specific logic constraint (.NCF) file which Xilinx place and route uses to determine the timing constraints of the circuit. The "trace" program report provides static timing information and constraints applied for place and route. An abridged version of the "trace" report file is given in Table 5.2.

Timing Summary	
Timing Errors	0
Constraints coverage	90.5%
Number of Paths covered by constraints	20,980
Number of connections covered by constraints	5,195
Design Statistics	
Minimum period (Maximum Frequency)	17.281 ns (57.867 MHz)
Minimum input arrival time before clock	14.251 ns
Maximum output required time before clock	15.179 ns

Table 5.2 Summary of Verbose Timing Report generated by Trace utility

The Post synthesis timing analysis results can be viewed using the Xilinx Timing Analyzer tool. The tool can be used to determine if the timing constraints were met successfully. Figure 5.9 shows a screen view of the Timing Analyzer tool indicating all timing constraints were met for the design.

If the process indicates the presence of timing errors then the timing report is analyzed to determine which constraint was not met. The mapping process is re-run after placing stricter constraints on the net or signal, which failed to meet the constraint. Some signals or nets that are critical for the timing constraints to be satisfied can be manually mapped to higher speed interconnects available in the FPGA. This is done by setting attributes in the constraint file that force the synthesis tool to map the net to a particular type of interconnect.The maximum frequency obtained as a result of the timing analysis is found to be 57.867 MHz, which is close to the 60 MHz design constraint. However though the minimum period (17.281 ns) is 1.281 ns more than the intended 16 ns, the difference was not found to be large enough to create any timing errors in the design. The high value of maximum frequency was achieved by introducing constraints on the set-up and hold times of the signals. The constraints were set up to ensure up to 90% coverage of the entire design. Critical paths were associated with more stringent constraints to

yield better results in the mapping. The mapping process is repeated until all constraints are satisfied.

Figure 5.10 shows the timing summary of the final routed design. The timing analysis did not generate any timing errors.

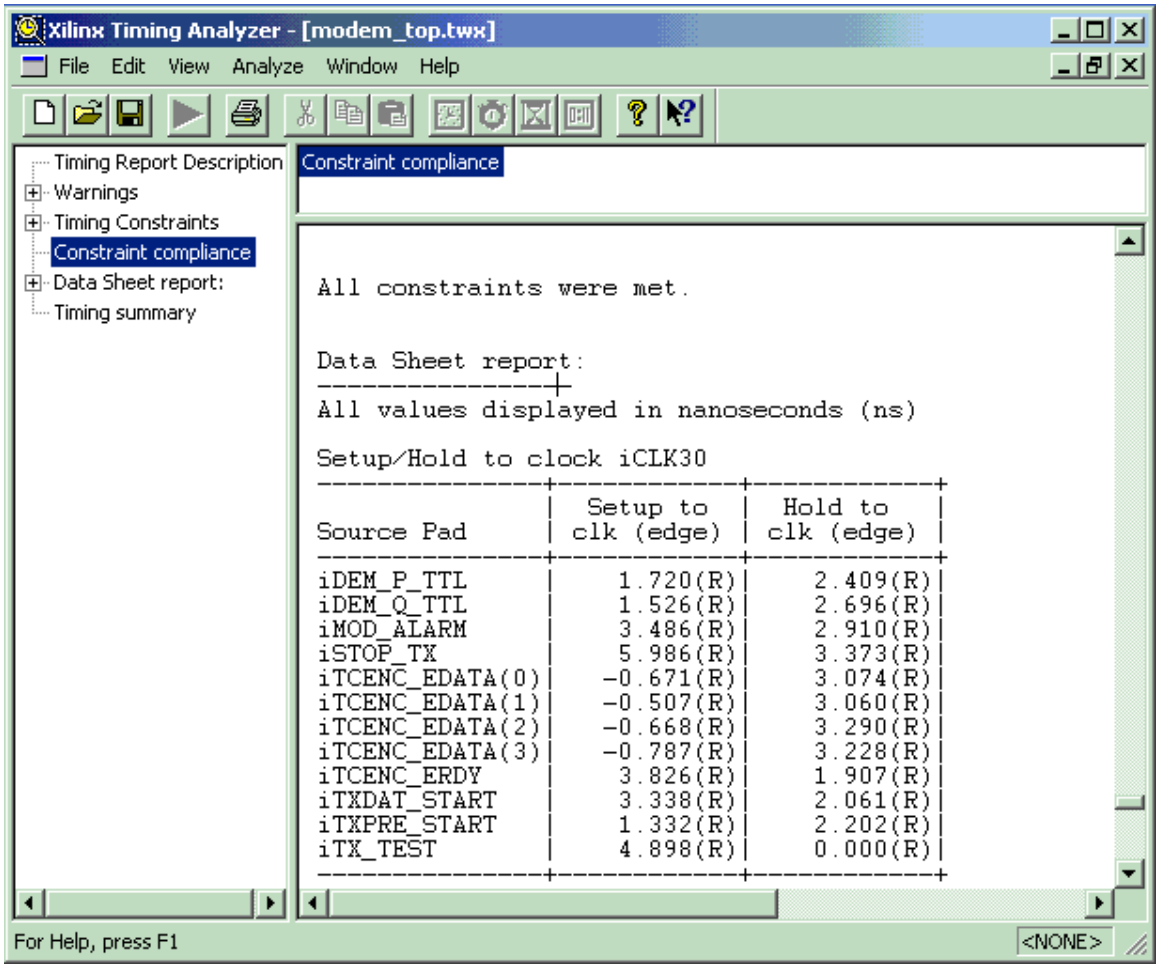


Figure 5.8 Xilinx Timing Analyzer Tool screen view - Constraint Compliance

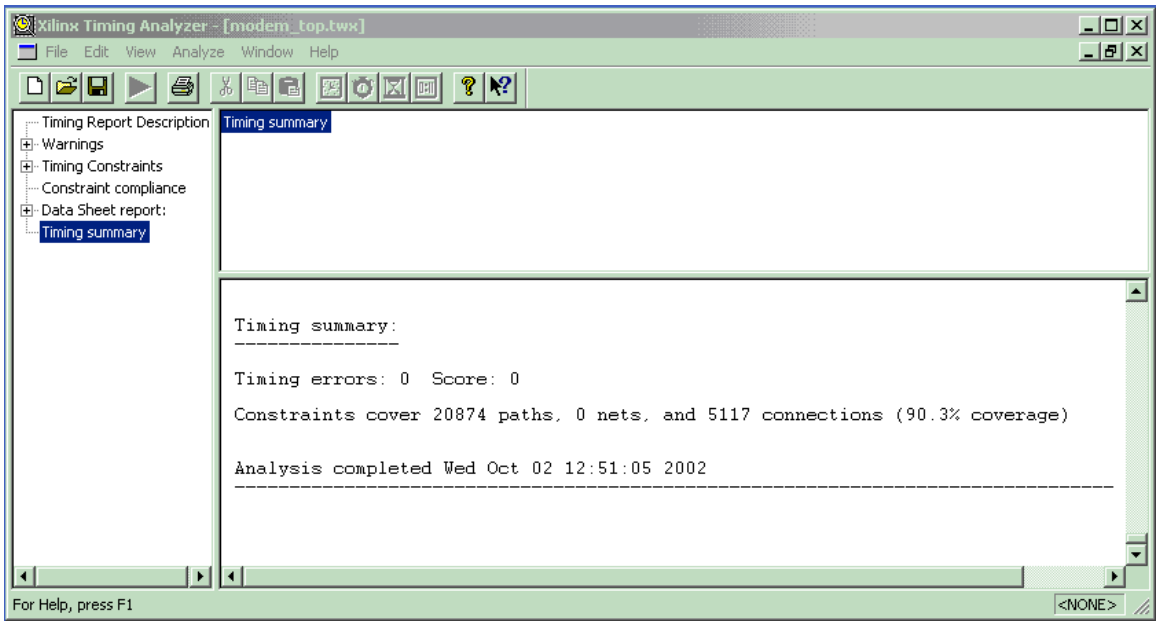


Figure 5.9 Xilinx Timing Analyzer screen view - Timing Errors Summary

5.2 Software Design Verification

Application software developed for the Gateway controller board also has to be tested. The Tornado tool offers several debugging options that can be used in the verification process. VisionWARE tools that are part of the Tornado suite of tools can be used to develop software that directly interfaces with hardware. The diagnostic tool accelerates and simplifies the testing process by providing generic as well as processor-specific tests.

5.3 Loop-back Testing

The Gateway controller is designed to support testing using a loop-back test feature, which allows the output of the QPSK Modulator to be directly input to the QPSK Demodulator – thus bypassing the radio links. This loop-back test feature can be used to test the operations of the Gateway controller independent of the system and isolate design faults in the Gateway controller from the rest of the system. The loop-back tests allow the testing of the entire transmit and receive paths independent of the physical radio link.

5.4 System Integration and Trials

The Gateway controller is carefully designed to interface with the rest of the Disaster Response System components using standard I/O interfaces or protocols such as Ethernet, RS 232 serial port, Positive Emitter-coupled Logic (PECL) I/Os, etc. The interfaces to the Sounder are described in Appendix A. The I/O interfaces are described in detail in [22].

5.5 Summary

The Gateway controller logic was simulated and verified after each design step. After the synthesis of the logic the synthesis tools were used to verify that all timing constraints are satisfied. The next chapter provides some conclusions and areas for future work.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

This work focuses on the problem of simultaneously designing and implementing data link and media access wireless protocols. The challenge is that of rapidly translating the informal system requirements into a formal specification that can be used to prove the effectiveness of the protocol using a prototype hardware and software. Ultimately, the specifications must be mapped onto an implementation, and it is at this point that the work in this dissertation aims to add value. The technical challenge is primarily that of balancing various conflicting or competing design objectives.

We propose a configurable architecture for embedded communication systems. The proposed architecture is a memory-centric hybrid architecture that employs a communications processor SOC platform and a FPGA based co-processor. The proposed configurable architecture is modular and can be classified into three distinct types of components: Processing Elements, Functional Units and Memory Elements. The Functional Units perform various data processing operations along transmit and receive data paths. The Processing Elements perform data transfers between data processing stages or functional units. Memory Elements act as buffers for processing elements moving data between Functional units. The control path elements consist of timing, status and control elements.

The FPGA logic was then mapped to combination logic blocks on FPGAs using a synthesis tool. The implemented FPGA logic was found to satisfy all timing constraints based on the detailed timing reports generated by the synthesis tool. The configurable prototype architecture allowed the use of a combination of FPGA IP cores and custom ASICs. Since some components like the Turbo Product Code CODECs did not have

readily available IP cores, commercial ASICs were used. The modular nature of the architecture is suitable for a component-based bottom-up design as in the case of Platform-based Design. The architecture also allows for rapid prototyping by allowing reuse of components and IP cores. The use of a Platform-based design methodology also helped to reduce design time by increasing design reuse. The hardware platform was simulated using behavioral VHDL simulators. Based on the simulation the design was found to conform to the design specifications and satisfy the timing criteria.

6.2 Contributions

A configurable architecture for prototyping embedded wireless communication systems was proposed. A hardware platform was designed to verify the design of the LMDS Gateway controller for disaster response networks. The scope of this work extended from developing specifications to component-level board design. The schematic and the printed circuit board (PCB) layout design were performed by external contractors. The VHDL code for synthesis of the FPGA logic was also developed and tested. Algorithms were developed for the application software to implement the TDMA MAC protocol.

6.3 Current Status and Future work

At the time of writing this thesis, the hardware platform design and verification has been completed. The software platform is still in the conception phase and is being developed at the Center for Wireless Telecommunications. System tests and trials will be conducted to verify the effectiveness of the TDMA medium access and lower level protocols.

One of the interesting avenues for future work is to use the new generation of "platform FPGAs" that have a built-in processor cores such as the Virtex II series which have programmable logic gates and a processor on the same silicon chip. Another interesting design avenue is to use IP cores instead of the various external ASICs as when

they become available. This would open up interesting areas of research in re-configurable computing. For example, a Reed-Solomon IP core can be dynamically re-configured to change the coding level and may result in a more efficient solution. This would also aid in the study of the adaptive protocols and techniques. The dynamically re-configurable property may also be useful in the context of rapid deployment. The system can use configurable IP network interfaces that can dynamically reconfigure the network interfaces to be compatible to those on the ground or to suit the needs of different response agencies.

Another logical extension of the research would be the development of a system platform, i.e. a combination of a hardware platform and software platform that allows for hardware-software co-simulation. New tools that allow hardware simulators and software emulators to co-operate, for example, by creating and allowing access to read and modify virtual "memory regions" can simplify the design verification aspects which continues to be a huge challenge for complex wireless system designs.

Bibliography

[1] H. Bölcskei, A.J. Paulraj, K. V. S. Hari, R. U. Nabar and W. Lu, "Fixed Broadband Wireless Access: State of the Art, Challenges, and Future Directions," *IEEE Communications Magazine*, Vol. 39, Issue 1, January 2001, pp. 100-108.

[2] M. J. Bass and C. M. Christenson, "The Future of the Microprocessor Business," *IEEE Spectrum*, Vol.39, Issue 4, April 2002, pp. 34-39.

[3] S. F. Midkiff and C. W. Bostian, "Rapidly Deployable Broadband Wireless Communications for Emergency Management," *National Digital Government Research Conference*, May 2001.

[4] C. W. Bostian and S. F. Midkiff, "Demonstrating Rapidly Deployable Broadband Wireless Communications for Emergency Management," *National Digital Government Research Conference*, May 2002.

[5] S. F. Midkiff and C.W. Bostian, "Rapidly-Deployable Broadband Wireless Networks for Disaster and Emergency Response," *The First IEEE Workshop on Disaster Recovery Networks* (DIREN '02), June 2002.

[6] C. J. Rieser, "Design and Implementation of a Sampling Swept Time Delay Short Pulse (SSTDSP) Wireless Channel Sounder for LMDS," Master's Thesis, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, 2001.

[7] T. J. Eshler, "Adaptive Protocols to Improve TCP/IP Performance in an LMDS Network using a Broadband Channel Sounder", Master's Thesis, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, 2002.

[8] A. Ferrari, A. Sangiovanni-Vincentelli, "System Design: Traditional Concepts and New Paradigms," *Intl. Conf. on Computer Design* (ICCD), pp. 2-12, 1999.

[9] K. Kuetzer, A. R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform based Design", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol.19, No.12, pp. 1523-1543, December 2000.

[10] J. M. Rabaey, M. Potknojak, F. Koushanfar, Suet-Fei Li, T. Tuan "Challenges and Opportunities in Broadband and Wireless Communication Designs," *IEEE/ACM Conf on Computer Aided Design* (ICCAD), pp.76-82, 2000.

[11] M. Iliopoulos, T. Antonakopoulos, " A Methodology of Implementing Medium Access Protocols Using a General Parameterized Architecture," *Proceedings of the 11th International Workshop on Rapid System Prototyping* (RSP), pp.2-7, 2000.

[12] S.K. Knapp, "Using Programmable Logic to Accelerate DSP Functions", Xilinx Inc., 1996.

[13] Motorola Inc., "MPC8255 Hardware Specifications", rev. 0.3, May 2002. http://www.motorola.com

[14] Xilinx Inc., "Virtex 2.5V Field Programmable Gate Arrays", Product Specifications v2.5, April 2001. http://www.xilinx.com/

[15] V. Subramanian, J. G. Tront, S. F. Midkiff, C. W. Bostian, "A Configurable Architecture for High Speed Communication Systems", *Military and Aerospace Applications of Programmable Logic Devices (MAPLD) International Conference*, Vol. 3, pp E11 1-9, Sept. 2002.

[16] Co-Optic, Inc., "Coic5130A: Programmable Reed-Solomon Encoder and Decoder Specifications", Device Specification, 1998.

[17] Advanced Hardware Architectures, Inc., "AHA4540Astro OC3 155 Mbps Tyrbo Product Code Encoder/Decoder", Product Specification, December 2001.

[18] Lockheed Martin Global Telecommunications, "Virginia Tech 120 Mbps Modem Interface Card", Interface Control Document – MIC to Modem, 00093 Rev 01, April 11, 2001

[19] Wind River Systems, "VxWorks 5.4 Programmer's Guide", Edition 1, 25 March 1999.

[20] Wind River Systems, "Tornado 2.0 User's Guide", Edition 1, 9 April 1999.

[21] Wind River Systems, " Tornado BSP Developer's Kit for VxWorks User's Guide", Edition 1, 9 November 1999.

[22] Visvanathan Subramanian, "Virginia Tech 120 Mps LMDS Gateway Controller", Design Specification Document, July 2002.

[23] Synopsys Inc., Synopsys Design Compiler, http://www.synopsys.com

[24] Synplicity Inc., Synplify version 7.0, http://www.synplicity.com

[25] Xilinx Inc., Xilinx Foundation ISE version 4.2i, http://www.xilinx.com

APPENDIX A

A. Gateway-Sounder Interface

A.1 Sounder Interface

This document discusses the interface between the Modem controller and Sounder units at the hub and remotes.

A.2 Modem – Sounder Synchronization

All the Modem controllers maintain synchronization information regarding the super-frame and sounder gap. The hub unit maintains the timing reference regarding the Super-frame commencement and transmits a special 32-bit "Sync Preamble" sequence. When the remotes detect this sequence, they update their timing references to align with that of the hub unit, thus maintaining synchronization.

The sounder is composed of a transmitter and receiver unit. The sounder transmitter unit is present at the hub and the sounder receiver unit is present at the remote. At present the sounder is being operated in a stand-alone fashion. The sounder operation is started and stopped manually through the PC interface at the sounder receiver unit. But on integration with the system the Sounder must operate only for the duration of the sounder gap so as to not interfere with the data transmission. For this to happen the sounder must also be synchronized with the rest of the system. The sounder transmitter and receiver units must be given an indication of the sounder gap beginning and end times by the modem controllers at the hub and remotes respectively.

A.3 Interface between the Sounder Transmitter and Hub Modem Controller.

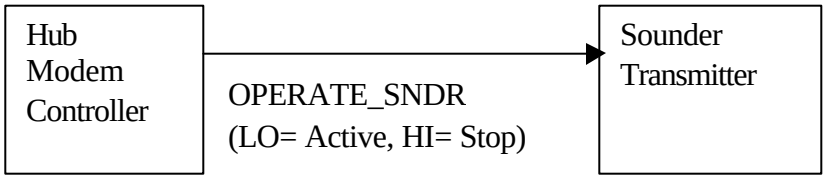


Figure A.1 Sounder Transmitter and Hub Modem Controller Interface.

The interface will be a simple level sensitive signal originating from the modem. The sounder can transmit as long as the signal is low but must remain silent if the signal is high. Many options, listed in Table A.1, were considered and a PECL interface was finally selected. The signal details are yet to be determined.

I/O Type	Benefits
LVTTL/CMOS	Standard I/O
ECL/PECL	Noise immunity for high freq, good drive capability
Opto-Isolator	Very high noise immunity

Table A.1 Sounder transmitter interface options

A.4 Interface between the Sounder Receiver and Remote Gateway.

In the current manual configuration, the sounder communicates with the host PC though a parallel port. The sounder board also has a serial port and a host port. Both these interfaces are designed to connect the DSP to microprocessors and other peripherals. The controller could use one of these interfaces to communicate with the sounder. The exact nature of the PC-sounder communication needs to be studied before we can decide on the interface.

A possible solution is to move a part of the sounder control software on the PC to the modem controller. In this configuration the sounder would be connected to the PC through the parallel port and to the modem controller through either the serial or the host port. The modem controller would signal the sounder to start collecting samples at the beginning of the sounder gap and provide any configuration details if necessary. This configuration may be changed if necessary through the PC-Modem controller interface. When the end of the sounder gap is signaled by the Modem, the sounder indicates to the PC that it has collected samples and the PC can then collect the samples through the parallel port. This is one possible scenario provided that the requisite Sounder interfaces are available and the nature of the PC-Sounder communications allows this.

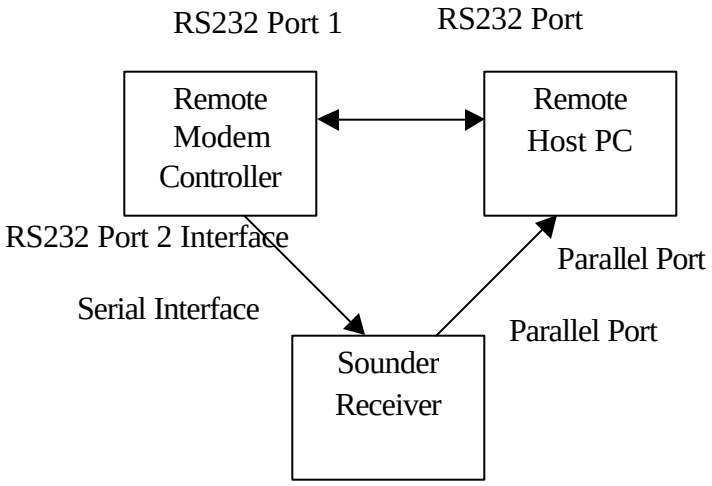


Figure A.2 Sounder Receiver and Hub Modem Controller Interface.

APPENDIX B

B. FPGA Logic Blocks

B.1 FPGA Target

The board consists of a 680 pin Xilinx Virtex FPGA (XCV 600). The Virtex family was chosen for its higher pin counts and compatibility with 5V outputs. The FPGA logic was designed using synthesizable VHDL. Two different environments were used to for design verification and synthesis respectively. For design verification a Test bench wrapper was written which had instances of the Modem logic and behavioral models of the processor, memory and FEC ASIC s. This design was functionally verified using the Synopsys VHDL Design Compiler.

B.2 FPGA Design Environment

For Synthesis of the design, the VHDL files were Mapped using Synplicity 7 and then Xilinx Foundation Tools ver 2.1 is used for Place and Route and to create the FPGA Configuration files.

B.3 FPGA Logic

The VHDL design entities in the transmit path namely, DMA1, DMA2, Modulator Interface (MODIF), Address Generator for DMA1 (M1ADDRGEN) are organised under Transmit top block (TX_TOP). Similarly the Receive Top block (RX_TOP) contains DMA3 (DMA3,DMA3o), DMA4, and the Demodulator Interface which constitute the receive path. In addition the TX_TOP and RX_TOP blocks each contain a 4Kx8 Dual-port SRAM Core EDIF file created using the Xilinx Coregen utility. The Modem Top block also contains a bus arbiter (M1BUSARB) to arbitrate access to the external DP SRAM's "left" port and a Microprocessor Interface (MPCIF) for communication between the FPGA and processor. Each sub-block is explained in greater detail in the following sub-sections.

B.3.1 Transmit Path Modules

The transmit path refers to the data path from Ethernet to the Modulator. As mentioned earlier the Transmit path consists of DMA 1, DMA2 and the Modulator Interface.

B.3.1.1 DMA1

The DMA1 is responsible for providing data from the external dual-port memory to the Reed Solomon Encoder for encoding and then storing the encoded data in the transmit side internal dual-port SRAM. The DMA1 reads the MAC packet in 180-200 byte sessions depending on the RS Coding level. The Reed Solomon encoder performs nine such sessions to always obtain 1800 bytes of RS Encoded data.

The processor provides the beginning address of the data packet and the RS Coding level that is being used through the general-purpose I/O ports. Once the appropriate registers have been written the values for the Beginning address (DMA1_BEGADDR) and Coding level (RSMODE), the processor asserts the DMA1_START signal. The DMA1 then initializes the Reed Solomon Encoder based on the value in the RSMODE register. Once the initialization steps are complete, the data transfer can begin. The DMA1 then requests access to the left port of the Dual-port SRAM from the arbiter by asserting the DMA1_BUSREQ signal. After it receives a DMA1_BUSGNT from the arbiter the DMA1 has complete access over the memory port. It begins to read data in bytes and places them at the data input of the RS Encoder. The transfers are synchronous to the 40 MHz clock RSENC_CLK. The data is available at the output of the encoder after a latency of 3 clock cycles and is indicated by the encoder by asserting the RSENC_RDY signal. The DMA1 maintains three counters CNT_TEMP, PAR_CNT and PKT_CNT to keep track of the number of data bytes, parity bytes and sessions respectively. The state transition diagram for the DMA1 is shown in Figure C.1.

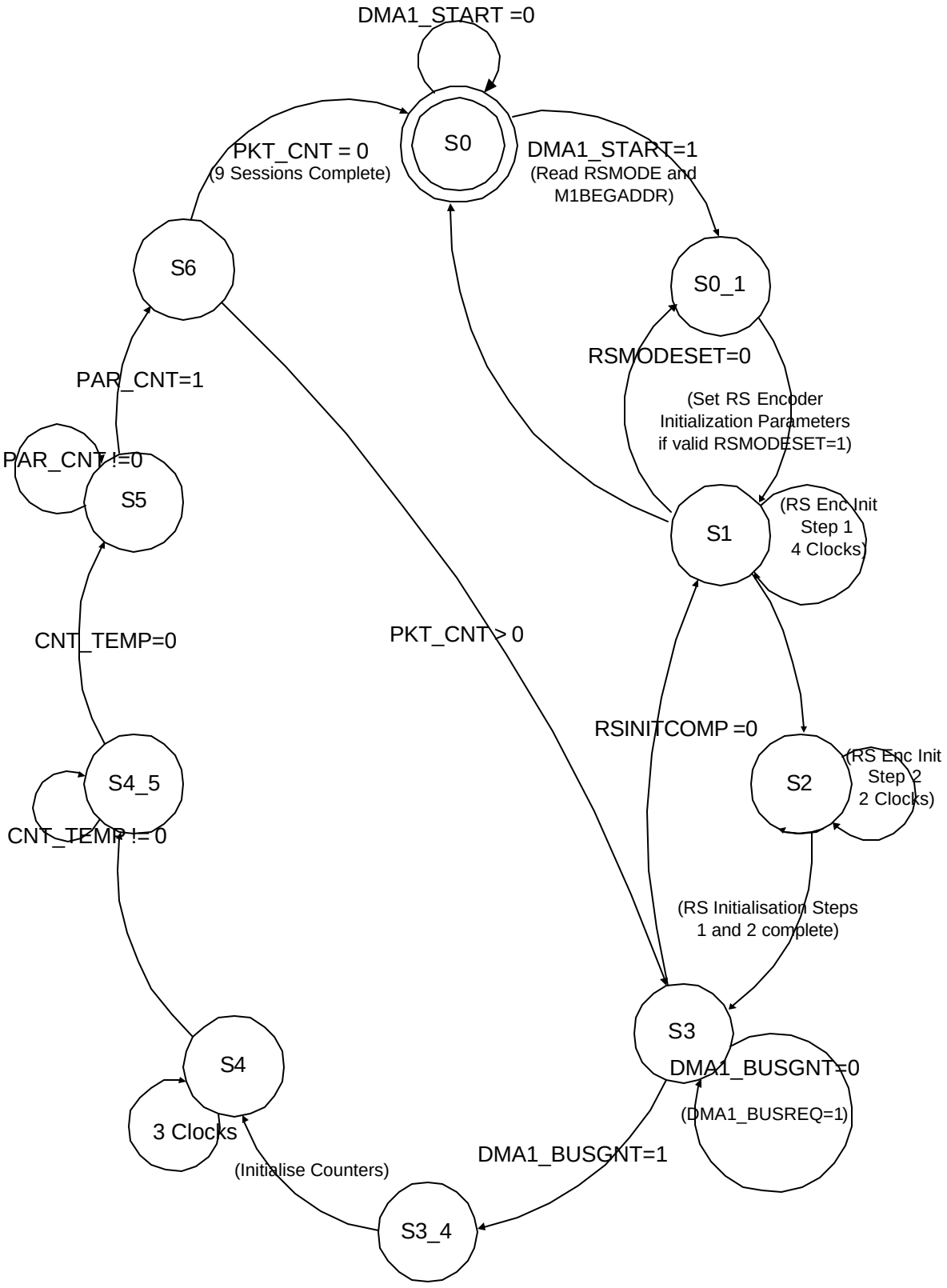


Figure B.1 DMA1 State Transition Diagram

B.3.1.2 DMA2

DMA2 transfers data from the internal dual-port SRAM to the Turbo Encoder input. The processor initializes the Turbo Encoder at start up. Once the Turbo Codec is ready to receive the input, it asserts TCENC_UACPT. The DMA2 indicates the beginning of a transfer by asserting TCENC_URDY during the first byte of the transfer. The number of bytes to be transferred is 1816 bytes that include the RS encoded header and data. The TC_IN_COUNT counter keeps track of the number of bytes transferred.

The State transition diagram for DMA2 is shown in Fig. C.2.

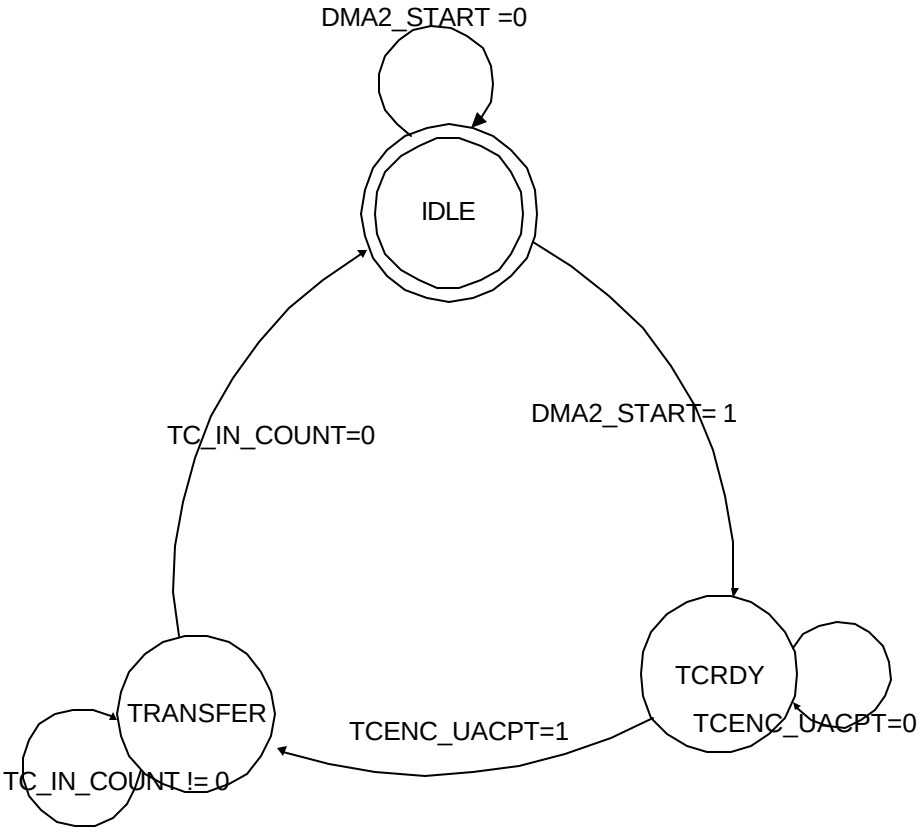


Figure B.2 DMA2 State Transition Diagram

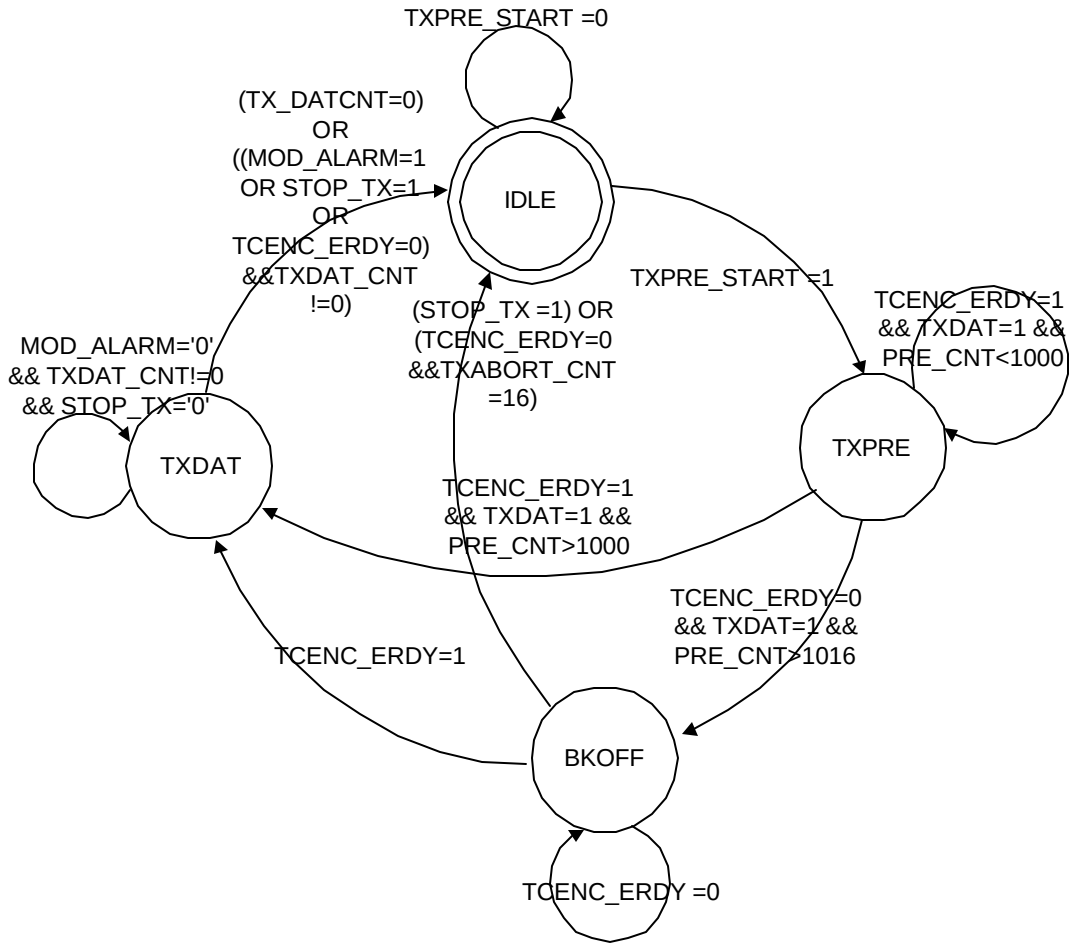


Figure B.3 State transition diagram for Modulator Interface

B.3.1.3 Modulator Interface (MODIF)

The processor keeps track of the slot times and gives the indication to the FPGA logic as to when to start or stop transmission. The assertion of the TX_PRE signal indicates that the preamble transmission must begin. The Modulator interface outputs alternate 0s and 1s for the preamble. A minimum of 1000 symbols and a maximum of 1016 symbols of preamble can be transmitted. If the MODIF does not receive a TX_DAT or TCENC_ERDY signal within 1016 symbol cycles of the TX_CLK then the transmission is aborted. The assertion of TX_DAT signal by the processor indicates that the data transmission should begin. At the end of the data transmission, TX_DONE is asserted to indicate successful transmission.

The processor can halt the transmission by asserting the STOP_TX signal. Alternately if there is an error in the Modulator indicated by MOD_ALARM then the transmission would stop. These conditions result in the assertion of TX_ERR signal indicating a transmit error condition. The State transition diagram for the Modulator interface module is shown in Fig. C.3.

B.3.2 Receive Path Modules

The receive path refers to the data path from the Demodulator to Ethernet. As mentioned earlier the receive path consists of DMA3, DMA3o, DMA4 and the Demodulator Interface.

B.3.2.1 Demodulator Interface (DEMODIF)

I and Q inputs are input directly to the Turbo decoder on the rising edge of TCENC_CCLK when TCENC_CRDY=1. Therefore in this case when the LISTEN signal is asserted the TCENC_CRDY is to be tied high. The turbo CODEC assumes every clock has valid data and determines the beginning of the frame by looking for Frame Sync sequences inserted by the turbo Encoder before transmission. TCENC_CACPT =1 indicates Buffer overflow which may have been caused by incorrect configuration.

B.3.2.2 DMA4

Decoded Data is output from the decoder on the rising edge of TCEDEC_DCLK when TCDEC_DACPT=1. TCEDEC_DACPT is to be tied high when LISTEN =1 indicating that the DMA4 is always ready. The turbo decoder asserts TCDEC_DRDY for valid data. TCDEC_DSTART and TCDEC_ DEND indicate start and end of data respectively. The decoded data is stored in the receive side internal dual-port SRAM.

TCDEC_DERR when asserted indicates an error that could not be corrected. At this point the processor can decide to drop the packet or if RS Encoding is turned on then it can continue processing it.

B.3.2.3 DMA3

DMA3 moves data from the internal dual-port SRAM to the inputs of the Reed Solomon decoder. The assertion DMA3_START signal causes the initialization of the RS Decoder for the header RS Code.DMA4 then reads the first 16 bytes and inputs it to the RS Decoder. It then waits for the information about the payload coding before proceeding to decode them. Once the information is received the RS Decoder is initialized the specific code and the payload is decoded. The State transition diagram for the DMA3 module is shown in Fig. C.4.

B.3.2.4 DMA3o

DMA3o transfers data at the output of the RS decoder to the external dual-port SRAM. When it receives the DMA3_START signal it asserts the DMA3_BUSREQ signal to indicate to the arbiter that it needs access to the external DP SRAM. Once it receives the DMA3_BUSGNT it can begin the transfer. The State transition diagram for the DMA3o module is shown in Fig. C.5.

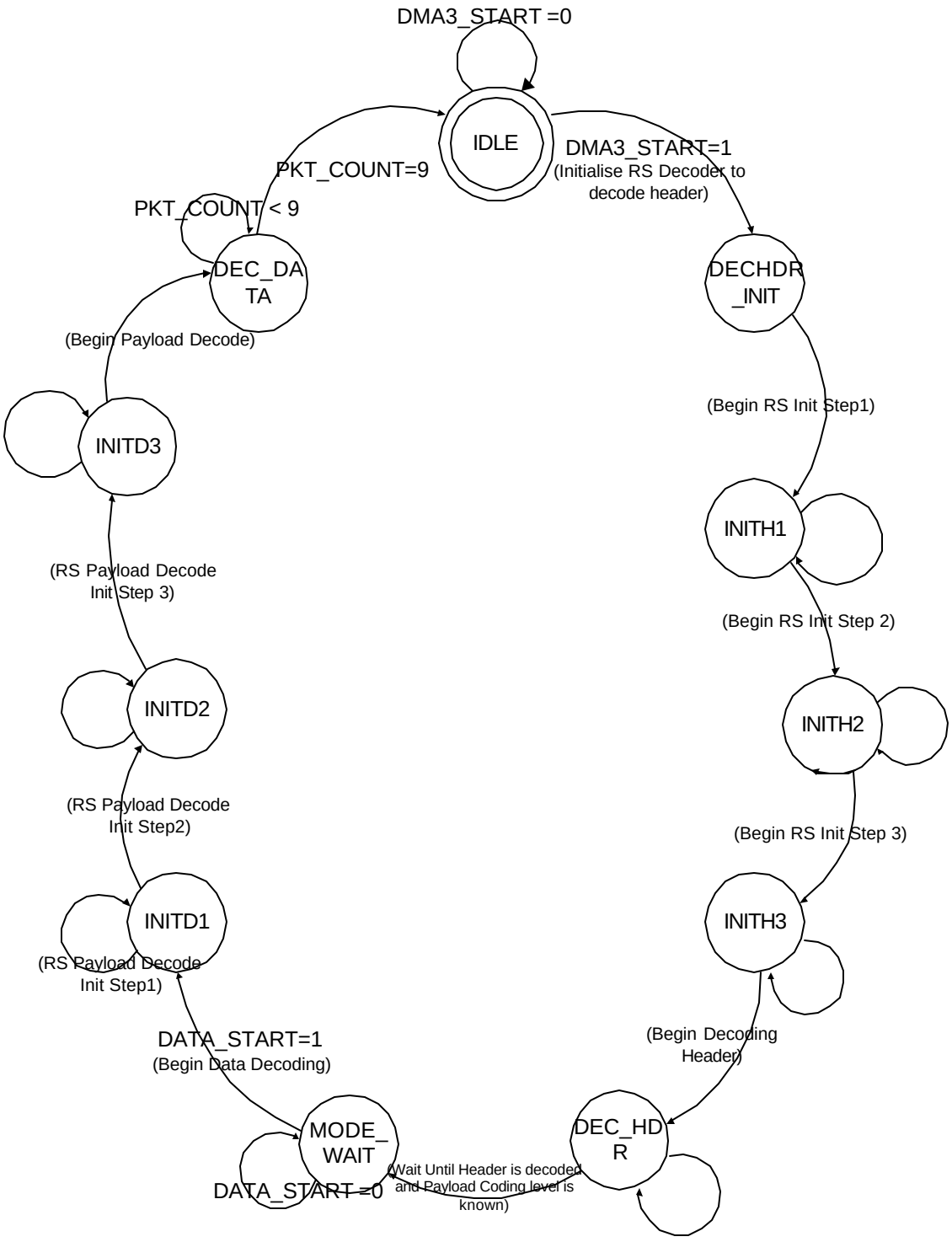


Figure B.4 DMA3 State Transition Diagram

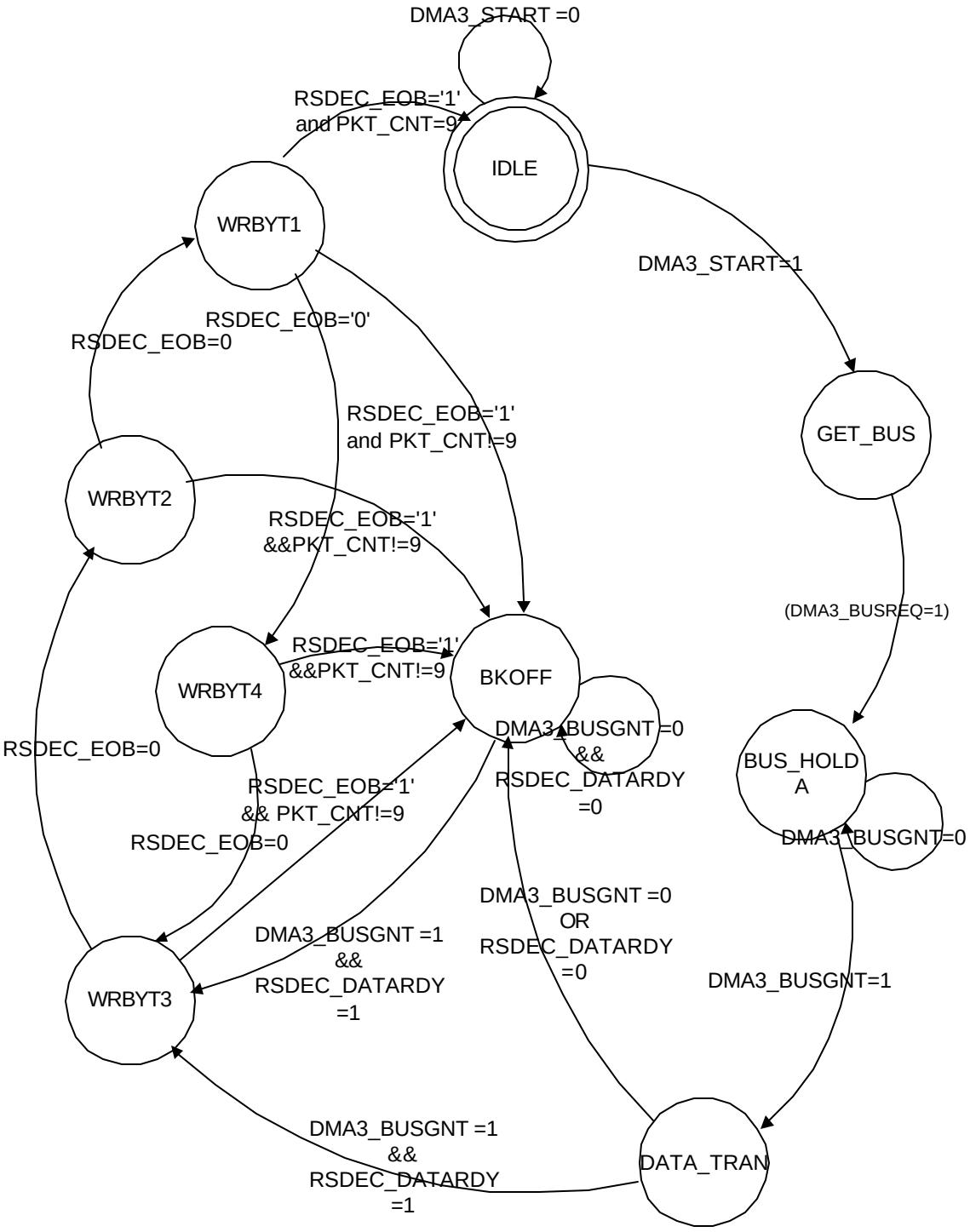


Figure B.5 DMA3o State Transition Diagram

B.3.3 External SRAM Port Arbiter

The External SRAM port arbiter implements a fair arbitration algorithm. When DMA1 and DMA3 request the bus at the same time then the person who held the bus last has the lower preference. Figure C.6 shows the state transition diagram for the Arbiter

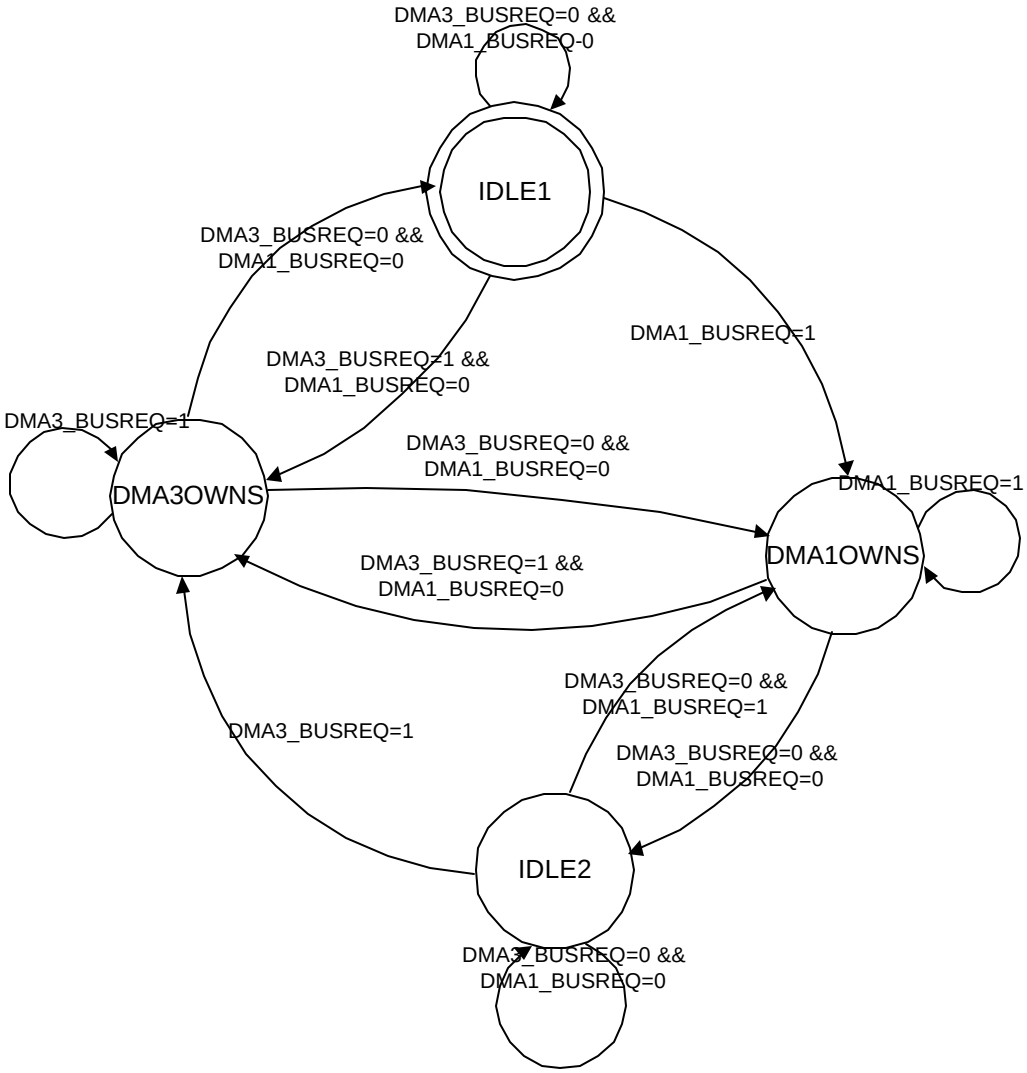


Figure B.6 Fair Bus Arbiter for External Dual-port Memory Port

B.3.4 Microprocessor Interface

The Microprocessor Interface consists of the Local Bus interface and the general purpose IO ports.

B.3.5 Timing Control

The timing control logic consists of a bit correlator to detect the Super Frame sync bits and timers to keep track of the transmission time slots.

Vita

Visvanathan (Vishu) Subramanian was born on December 6, 1978 in Hyderabad, India and was raised in Coimbatore, India. He received his Bachelor of Science degree with distinction in Electronics and Communication Engineering in May 2000 from PSG College of Technology, Coimbatore.

In the summer of 1999, he received an opportunity to work at the prestigious Indian Institute of Science (IISc) at Bangalore, India under the IMPACT Project – a joint project of the World Bank and Department of Electronics, India to encourage scientific research in undergraduate study. At IISc, as a member of the Network Analysis Group at the Center for Electronics Design and Technology, he developed "packet-sniffing" tools to perform LAN Traffic Analysis and also designed protocol drivers for Ethernet NIC cards. At PSG College of Technology, he served as the Secretary of the Astronomy Club from 1999 to 2000. He was also Editor-in-chief of the PSG Tech College Magazine and the PSG Tech Electronics and Communication Engineering Journal. After completing his Bachelor's degree, he joined Cognizant Technology Solutions Inc., as a System Analyst in June 2000.

He joined the Master of Science program in Electrical Engineering at the Bradley Department of Electrical Engineering at Virginia Tech in Fall of 2000. Visvanathan Subramanian has been pursuing research in digital broadband wireless system design at the Center for Wireless Telecommunications since January of 2001. As part of his MS Thesis, he investigated configurable architectures for wireless communication system design. He designed a LMDS wireless gateway controller for a rapidly deployable broadband wireless Disaster Response System. His research interests include configurable architectures, VLSI design and digital embedded system design for wireless communications. Visvanathan Subramanian is a student member of IEEE.