AFRL-IF-RS-TR-2003-258
Final Technical Report
November 2003

UNIVERSAL NETWORK ACCESS SYSTEM

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J180

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-258 has been reviewed and is approved for publication.

APPROVED: /s/
DANIEL J. HAGUE
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 2003	**Final** **Aug 98 – Jun 02**

4. TITLE AND SUBTITLE

UNIVERSAL NETWORK ACCESS SYSTEM

5. FUNDING NUMBERS

C - F30602-98-C-0218
PE - 62110E
PR - G300
TA - 00
WU - 01

6. AUTHOR(S)

Kirk Boyer and Bob Davies

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Tektronix Federal Systems, Inc.
P.O. Box 4490 M/S 50-FSI
Beaverton OR 97076-4490

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2003-258

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Daniel J. Hague/IFGA/(315) 330-1885/Daniel.Hague@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

Develop and demonstrate a Universal Network Access engine using an innovative bit-rate adaptive demultiplexer/multiplexer, a reconfigurable physical layer protocol engine, and a reconfigurable cell/packet engine. FUNCTION: This work supports Air Force C3I Communications Networks. OPPORTUNITY: The effort will provide the network equivalent of a power adapter/extension cord for flexibility interconnecting a wide range of broadband information sources and local networks over the next generation internet. CONTRIBUTION: The capability to rapidly deploy high bandwidth, interoperable network which integrate information from different sources.

14. SUBJECT TERMS

Bit-Rate Adaptive Network Demultiplexer, Multiplexer, High Bandwidth Network Deployment, Interoperable Networks

15. NUMBER OF PAGES

156

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE of CONTENTS

S.

List of Figures

List of Tables

Abstract

This project developed flexible, software-configurable network interfaces and protocol processing engines. Bit-rate adaptability and protocol reconfigurability were realized by combining a custom ASIC, a Physical Layer Processor and a Cell/Packet Engine in a card-level Universal Network Access Engine (UNAE) that is reconfigured through software commands. Mezzanine cards adapt to various physical media. Protocol- and format-processing engines were developed and tested for OC-48 Packet over SONET (POS), Gigabit Ethernet, HD video and SD video, running on the same HW cards, with only SW reconfiguration. In combination, UNAE cards can select from a library of protocols and translate between them at several different data rates. The system communicates directly with other attached network elements such as to routers, using PPP/LCP for POS and auto negotiation for Gigabit Ethernet. The system demonstrated transport of uncompressed HD video (SMPTE 292) data at 1.5Gbps over an IP wide area network from Seattle to Washington DC over Internet2. A gateway interface was created between an OC-48 POS network and an Optical Label Switched Router (OLSR). IP packet header information was used to generate labels that instruct the OLSR how to switch each packet. The UNAS receives optical packets from the Optical Label Switched Network and recovers the data. The concept, architecture and engine elements have been proven in a series of system demonstrations that show the power and flexibility of the approach.

1 Summary

Much effort has been focused on building a Next Generation Internet infrastructure - to provide millions of users with gigabits per second bandwidth on demand, increased network automation and improved security, all at greatly reduced cost to the user. Prior to the Universal Network Access System (UNAS) project, much effort had been focused at addressing very high bandwidth network pathways and switching. Less progress had been made on technologies needed to provide transparent, service-independent, high bandwidth network access. This project has focused on providing flexible, software configurable, network interface adaptors which will be useful for a wide variety of applications.

The strategy was to pursue an approach combining elements supporting bit-rate adaptability and protocol reconfigurability into a system architecture on which to build a variety of network adaptors and interfaces. Three key functional blocks to be developed included a bit-rate agile DeMux/Mux for adapting to a variety of network speeds and signal sources, a Physical Layer Protocol Engine and a Cell/Packet Engine. The Physical Layer Protocol Engine provides a flexible means of handling physical layer and transport functions including termination of physical links, scrambling/descrambling, bit and Byte alignment and framing and recovery of packetized or formatted data. The Cell/Packet Engine does processing at the packet or cell level, dealing with verifying integrity of data payloads, timing synchronization between user and network clock domains, and processing of user data. This is not only a flexible key to interfacing between user data and the network, but it may provide powerful support for determining quality of service.

The proposed elements were combined into an architecture; and a series of card-level Universal Network Access Engines (UNAEs) were developed. These share common hardware and are reconfigured through software commands and an embedded operating system. In meeting the objectives above, the requirement to adapt to various physical media attachments is realized through a small set of mezzanine cards that each attach to a motherboard to form the UNAE element.

These engines, when integrated into a simple multicard system, have the following benefits:

They provide the ability to rapidly and inexpensively deploy edge elements for a variety of networks using a base set of hardware with software configurability (taking only a few seconds to configure).

Engines provide flexibility in providing scalable network access for a variety of different services and data sources. This was demonstrated by providing interfaces from both standard definition (SD) and high definition (HD) digital video sources and two common types of packet based, high bandwidth transport networks (Gigabit Ethernet and OC-48 Packet over SONET).

They lay the groundwork for the development and testing of new protocols. A simple example of this is the label-filtering protocol that was co-developed with University of

California at Davis, where a custom label was created from IP header information and sent in a reduced bit rate stream to an optical label processor to generate optical subcarrier multiplexed labels for an all-optical packet routing switch. There is a much richer set of custom protocols that these engines will allow - for applications focused on security, simplicity, or bit efficiency – that were not addressed in this program.

This program undertook to research architectures and methodologies needed to provide a highly flexible approach to realizing network access and payload processing elements. An architecture was developed that would accommodate common elements of protocols in significant commercial use, yet would provide the flexibility to allow development and testing of customized and proprietary protocols. A combination of dedicated integrated circuits (ASICs) and high performance programmable logic were chosen to comprise the engines. Design and development tools were surveyed and selected. The program became a testing ground for balancing the ease of use of tools with higher levels of abstraction and design automation, against the high performance demanded of the engine elements for high bandwidth protocol processing. Protocol Compiler and VERA from Synopsys were selected for higher level logic design and verification tools, respectively. Verilog took longer than Protocol Compiler for initial design; but the design approach allowed more direct control of performance, showing that for these very high performance designs, the Verilog is superior to Protocol Compiler.

The approach of using a mezzanine board for physical media attachment and the serialization/deserialization and clocking functions was very powerful. The key element of the mezzanine was to be the bit-rate adaptive DeMux/Mux (BRAD), which would provide bit-rate agile transmitter and receiver clock circuits from 150Mbps to 3 Gbps, burst mode capture, and parallel word widths on the deserialized side. A Complementary Metal-Oxide (CMOS) feasibility study based on available processes and models suggested that the risk of using CMOS would be too high. A BRAD design based on the IBM 5HP SiGe process was developed and simulated. This design was discontinued due to resource issues. In its place a commercial clock and data recovery and DeMux/Mux chip was successfully designed into the mezzanine. All testing was done with the commercial chip. Without the BRAD, video interface mezzanines also needed custom chips to handle the video signals. This led to successful implementations that required separate optical network-, SD video- and HD video-mezzanines.

Interfaces and protocol- or format-processing engines were successfully realized and tested for OC-48 Packet over SONET (POS), Gigabit Ethernet, HD video and SD video. That these can all run on the same card with only a few-second SW configuration show the strength of the architecture and approach using programmable engine cards with physical layer mezzanines. In combination, these cards also showed the ability to accommodate generally formatted user data, the ability to quickly select and transmit from a library of protocols, the ability to function at several different data rates (bit-rate adaptability) and the ability to translate between different formats or protocols including SONET to/from Gigibit Ethernet, video mapped to/extracted from SONET or Gb Ethernet. Packet and network diagnostics were also demonstrated, through reporting of SONET alarms and link status indications but also through packet and payload

diagnostics within processing elements in the UNAE itself.

The system was used to communicate directly with other attached network elements over a wide area network, talk to routers through PPP/LCP for Packet over SONET and auto negotiate for Gigabit Ethernet. The packet filtering and negotiation for this was handled autonomously on the UNAE card under control of its embedded operating system. The system fully negotiates for link connection with commercial test equipment such as IXIA and Adtech packet testers.

The ability to set up links with other network elements and the UNAE multiprotocol mapping were combined to demonstrate for the first time the transport of uncompressed HD video (SMPTE 292) data at 1.5Gbps over a wide area IP packet based network. This was demonstrated on several occasions from Seattle to Denver and to Washington DC over the Internet2 backbone, with diagnostics for dropped and reordered packets obtained for up to 18 hours at a time. Besides showing for the first time that this could be done, this provided unprecedented validation that routers on the network could handle extremely large, single flows of real time traffic such as these.

A final example of protocol flexible performance using the UNAS is the collaborative development with University of California at Davis of a gateway interface between an OC-48 POS network and the Client Interface for an Optical Label Switched Router (OLSR). IP packets were extracted from the OC-48 stream, the packet header information was used to generate a label that instructs the OLSR how to switch that packet, and the label and original packet data are forwarded to a optical label generation module. Customization of the stream such as encapsulation using HDLC for data scrambling and appending a preamble are also programmed into the data processing in the UNAE path. The UNAS also receives optical data from the Optical Label Switched Network and recovers the packets and data. This is another unprecedented demonstration, that of a completely custom and extendable interface between a conventional POS network and an Optical Label Switched Network. The architecture of the UNAE can be extended with the OLSR to other advanced developments such as generation of jumbo packets in concatenated streams.

The program did have some setbacks, including the inability to complete the BRAD within the program resources. This chip would have been one of the first burst mode receivers to run at 2.5Gbps. This would have eliminated the need for the cumbersome preambles in the OLSR client interface. The BRAD would allow a single mezzanine board to be used for all the protocols and formats addressed in the program, using a multiplex selector between a fiber and coax (video) inputs.

One of the biggest design challenges was to accommodate multiple clocks (not multiples of each other) such as SONET and Gigabit Ethernet. Having the BRAD would have addressed this too. However, in the implementation without the BRAD, the performance was limited by the components used. Failure for the vendor to successfully build oscillators within spec after our mezzanine redesign limited use of the system to run in “loop-time” (deriving the transmit clock from the received clock) rather than “local-time, where an on-board oscillator provides it. This did not impact any of the demonstrations of

features but limits the applications unless a suitable oscillator is added.

Use of Protocol Compiler tool was considered to have added some time to the full design task, since the performance demanded of the programmable logic by this program required careful redesign of each module to assure consistent performance in the integrated designs.

The intent of the research has been successfully carried out, however. A multirate and multiprotocol system, based on software configurable, hardware accelerated processing engines has provided a series of system demonstrations that show the power and flexibility of the concept, architecture and system approach.

2 Introduction

2.1 Purpose

Building the Next Generation Internet (NGI) capable of providing millions of users with gigabits-per-second bandwidth on demand has required breakthroughs in network access, transport, multiplexing and switching. Prior to this project, progress in high speed time division multiplexing, dense wavelength division multiplexing, and optical switching technologies had been addressing the needs for high bandwidth pathways and switching. Much less progress had been made, however, in access technologies to provide the network adapters required to achieve transparent, service-independent network access. This research has been directed towards realizing this network access adapter function.

Figure 1 shows several methods of connecting into the NGI core network through terminal adapters and edge devices. Commercial and DoD users who need to transport a variety of data types, including archived material as well as realtime data from satellites, motion imagery and sensor data, must first find a means to map their asynchronous, often

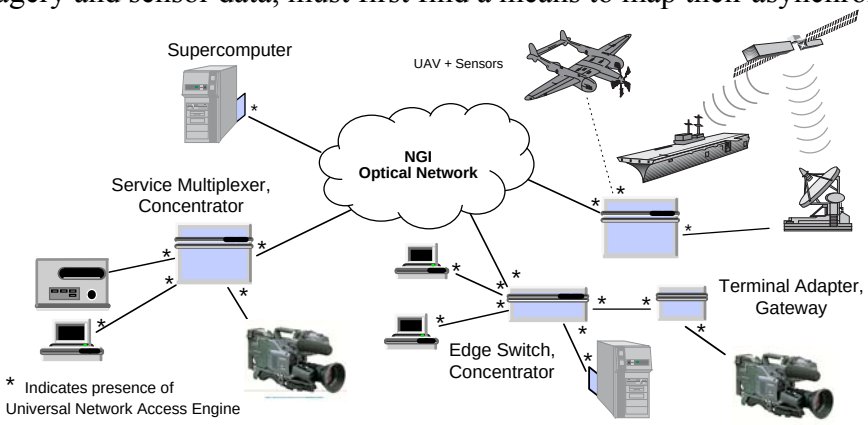


Figure 1 Use of Universal Network Access Engine in NGI Applications

proprietary I/O signals on to a standard network interface. In the process, they may need to wade through numerous protocol layers and mappings, e.g. IP to ATM adaptation layer to ATM physical layer to SONET. It is often difficult to build applications to take advantage of current high speed network testbeds because researchers have no means of connecting high speed signals from devices such as servers, supercomputers, and motion image cameras into the SONET network access ports. The development of specialized access devices is often a time consuming and expensive proposition. Furthermore, even if a specialized terminal adapter exists for a specific data source, it was still considered very difficult to modify protocols to perform network testing or to optimize protocols for new switching and multiplexing technologies. It was partly to address this last issue that the Optical Data Service Interconnect (ODSI) Coalition and Optical Internetworking Forum (OIF) were established in 2000 to develop a powerful set of User to Network and Network to Network Interfaces and protocols to provide the signaling, automation and testing for connected dissimilar networks on a large scale.

Achieving bit-rate adaptability and protocol reconfigurability formed the original focus of this research. The concept of a Universal Network Access Engine in theory affords several advantages:

Rapid capability to deploy interoperable networks and build new applications. In defense applications, adaptability could play a significant role in facilitating the deployment and adaptation of battlespace networks. For example, the need to integrate satellite, archive, sensor and other information to achieve information superiority may require an enormous amount of "glue" infrastructure such as network adapters, protocol converters, and format converters, each accompanied by logistic and inventory support demands. Achieving network connectivity for new types of sensors and interoperability with existing international standards can require development or procurement of new special purpose sensor adapters and protocol converters. Access engines with dynamic software configurability can reduce the time required to build or upgrade information networks in the field. An access engine can be reused and reconfigured as a network building block to form network interface cards, protocol transcoders, device and service adapters, and perhaps be extended to multiplexers, concentrators, edge switches, etc., for different data types and networks.

Operation with streamlined protocols. The programmable engine, in conjunction with multiwavelength optical switching, might enable the DoD to take advantage of NGI technology and clear optical channel access by providing flexibility in fine tuning both the bit rates and protocols for a given application. Sufficient bit rate flexibility, for example, would enable users to bypass the ATM layer by matching network access rates to sensor data rates.

Reconfiguration of physical layer protocol under software control. This can be used to provide enhanced network security by programming custom protocols which can be periodically or randomly altered.

Flexibility in providing scalable network access for a variety of different services and data sources. The same engine can be reused to provide a SMPTE 259 digital video interface, or SMPTE 292M HDTV interface, for example, or could even be reprogrammed to extend a protocol to support transport of a data type for which no transmission protocol has yet been standardized. As this research started there was DoD interest in utilizing high resolution framing sensors, e.g. 1080 line 60 fps progressive scanned HDTV cameras and other high data rate sensors whose outputs exceed the data rate of existing standards such as SMPTE 292M.

Development and testing of new protocols. While SONET, ATM and IP perform many necessary functions (including providing a hierarchy of standardized access rates, per-link error detection and handling, service mapping and bit rate decoupling, statistical multiplexing, resource allocation, switching, routing and the ability to provide decentralized network management) they also perform many overlapping functions which may not be optimal for optical networks. For example, the use of larger or variable size cells with unique IP-like addressing might allow the merging of ATM and IP into a single more streamlined layer. Larger cell sizes might be better suited to optical switching

technologies. The programmable access engine will enable network architects to try out modified or new protocols such as these.

Network monitoring and Quality of Service testing. Signal generation and monitoring functions can be embedded in the access engine to measure network performance. For example, time stamps can be inserted into packets or cells to monitor network jitter, and reference data signals can be generated to assess end-to-end performance at the application level.

All optical switching fabrics have been touted as providing scalable, reliable transport, especially with the rapid development of DWDM and the prospect of wavelength provisioning in the fabric. Optical switching fabrics have usually been envisioned as circuit switches; but there is value in all optical fabrics that can switch at the packet level in real time. Proposed switches such as optical label switched routers (OLSR) when run in a packet switching mode don't always interoperate will with existing packet networks which are in extensive and growing use today. The system developed in the current program was seen as a unique vehicle to experiment with interfacing contemporary networks such as IP packet over SONET to prototype future all optical, OLSR fabrics.

Scope of Project

Tektronix has contributed to the NGI program by researching and developing a "Universal Network Access Engine" to provide research in network access technology for flexibly interconnecting a range of broadband information sources, appliances and local area networks. The scope of this program is to research methods for flexibly interconnecting a wide range of broadband information sources and local area networks over the Next Generation Internet (NGI). The primary objective has been to develop the architecture, components, and a prototype system to demonstrate a Universal Network Access Engine using a bit-rate flexible modular optical interface, a reconfigurable Physical Layer Protocol Engine, and a reconfigurable Cell/Packet Engine. The engine architecture is intended to enable reconfigurable remote network monitoring and network edge devices, which can flexibly interconnect broadband information sources and appliances over the NGI. The technology has been developed to provide groundwork for rapidly deployable and reconfigurable networks, interconnection of dissimilar network types, development and testing of new protocols, packet label or tag generation, network monitoring and quality of service (QoS) testing, and is intended to also be extensible to new approaches to data security.

The following elements were addressed in the work and in the following report:

- Universal Network Access System (UNAS) and ASIC/programmable logic Architecture were developed and simulated.
- Development of NGI components including system, board level designs, ASIC and FPGA designs, including software development to meet the requirements. Support of continuous data processing associated with protocols at 2.5Gbps and above demands that the designs run in real time at 78-155 MHz at whatever processing depth is required.

- Development of a UNAS evaluation and demonstration environment: This refers to internal lab-level demonstrations. Additional purchases of a router with 2.5Gbps ports and network signal generation and measurement equipment were made by Tektronix to support program activities.
- Integration of UNAS to provide operational verification and UNAS evaluation and network demonstrations. A first time network demonstration of HDTV transport over an IP based wide area network was performed.
- Development of a programmable gateway interface between an all optical packet switch and contemporary SONET network.

These elements encompass the work statement for the contract. Technology was developed to enable operation from 150 Mbps to 3 Gbps. An ASIC architecture was developed and a design partially completed to address autonomous bit rate agile clock and data recovery over this performance range. The program has resulted in demonstration and delivery of a prototype Universal Network Access System (UNAS).

2.2 Technology Approach

The Universal Network Access System consists of a platform, one or more board level Universal Network Access Engines and controlling software. The Universal Network Access Engine concept relies on the development and interdependence of three unique building blocks: 1) the Bit-rate Adaptive Demux/Mux, 2) the physical layer protocol engine, and 3) the Cell/Packet Engine. Network interface cards, terminal adapters, service multiplexers, concentrators and other network edge devices may conceptually be constructed using Access Engines.

The Bit-rate Adaptive Demux/Mux (BRAD) is intended primarily for fiber optics network applications. On the receive side, the BRAD accepts a serial data input stream, recovers clock and data, and demultiplexes it. On the transmit side, the BRAD accepts data from a parallel interface and multiplexes it into a serial output stream. The BRAD generates clocks for serial and parallel data outputs, and clocks the parallel input.

The receiver and transmitter operate independently from each other at any frequency from 150 MHz to 3 GHz. Multiplex and de-multiplex ratios (more recently termed serialization and deserialization, respectively) are independently programmable to multiple bit widths to better accommodate the range of clock frequencies. There is facility to estimate data rate of received signals of unknown protocol for setting the initial frequency of the clock recovery circuit. The clock and data recovery in the receiver will recover burst data of a known rate without losing any initial data while locking in.

The BRAD concept provides features not generally available in commercial transceiver components for optical communications. These include a burst mode which allows the receiver to instantly start acquiring data, without losing a single bit, when a new transmission starts, using a known protocol. Most receivers have phase locked loop (PLL) clock recovery circuits, which need a lock-in period before data can be acquired. Bit rate agile transmitter and receiver clock circuits may be programmed to any serial rate from 150 Mb/s to 3 Gb/s. Programmable parallel word widths of 8, 10, 16, 20 and 32 in

receiver and transmitter parallel-to-serial and serial-to-parallel converters better accommodate multiple protocols. Finally, a receiver transition counter helps estimate the bit rate of a received data of unknown protocol.

The BRAD can be seen to combine the functions of burst mode receiver, clock and data recovery (CDR) and serialization/deserialization (SERDES) in a single circuit. Additional information from a transition counter in the BRAD is required for the system to recognize an unknown bit rate and is a key step in recognizing unknown protocols. If a commercially available CDR and SERDES are used in place of the BRAD, some of these functions may be lost.

The physical layer protocol engine (PLPE) is tasked to provide compliant protocol processing for communications applications through adaptive reconfiguration.

The PLPE takes advantage of the large degree of commonality among many physical layers to achieve protocol adaptability. For example, most protocols tend to have some type of frame structure for header and payload composition, usually with byte or word quantization. They also generally incorporate parity and/or Cyclic Redundancy Code (CRC) checks, and some form of encoding or scrambling. Often, overhead insertion and extraction is required. Some protocols require calculations over a frame which impact the headers of the subsequent frame. Of the commonly used standard protocols, SONET/SDH is one of the most processing intensive. Gigabit-per-second processing demands extensive levels of pipelining, signal and clock routing resources, and synchronized control over those resources.

The Cell/Packet Engine, in conjunction with the physical layer protocol engine is designed for flexibly processing cell or packet layer protocols required for statistical multiplexing, routing and switching. The proposed engine will be programmable to accommodate variable length cells or packets. It will also be capable of processing header structures to support time stamps, or filter on IP source and destination addresses for example. In addition, since optical switching technologies may require novel techniques for extracting headers from cells, the engine must flexibly accommodate a wide range of structures, even at the bit level, for embedding header information within the cell. It is this engine that provides for service-independent access by mapping asynchronous user data streams onto fixed rate network pipes. In addition to helping map cells to and from the physical layer payload, this engine is responsible for cell buffering to accommodate network jitter, header processing and clock recovery for data streams. The output of the engine is a general purpose cell/packet level data interface which can be programmed, for example, to conform to the ATM UTOPIA interface and its extensions such as SATURN POS/PHY or SFI-3.

The Universal Network Access Engine (UNAE) is a card consisting of the Bit-rate Adaptive Demux/Mux, physical layer protocol engine, and Cell/Packet Engine programmable elements, an embedded controller, FLASH memory, RAM, and auxiliary components. The Universal Network Access System (UNAS) is comprised of one or more Access Engines, a card cage or platform, and software consisting of an embedded real time OS, drivers, user interface, configuration libraries, and modules for accessing,

controlling, reconfiguring, managing, and monitoring the system. The flexibility and modularity of this system enables the configuration of a variety of network access devices, including network interface cards, terminal adapters, protocol transformers, service multiplexers and concentrators. Figure 2 illustrates an access system with multiple engines configured as an access concentrator.

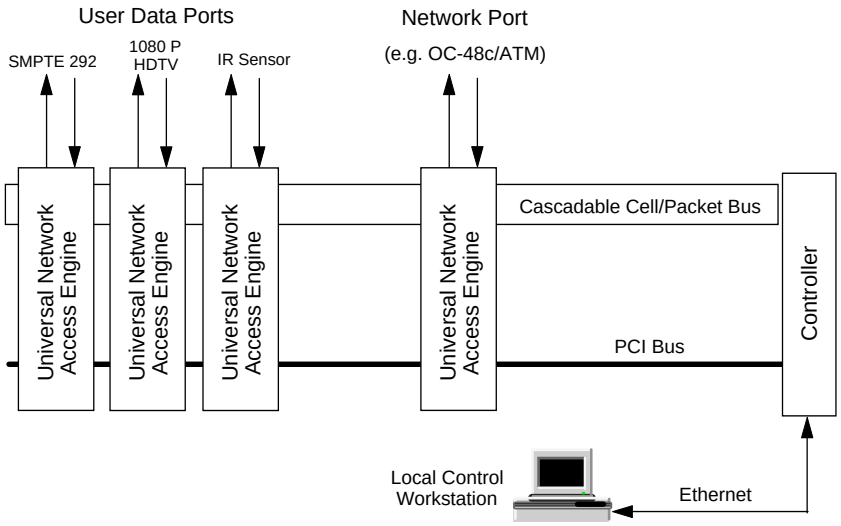


Figure 2 Use of Universal Network Access Engine in an Access ConcentratorPlan

This report covers the essential details of the investigation undertaken. Section 0 covers the system requirements and assumptions for this study. This is followed by discussion of methodology used to address the requirements, including selection of tools for hardware design and verification, selection of programmable logic elements, and identification of initial performance milestones. Development of the hardware and software architecture is included in this discussion.

Section 4 continues by addressing details of system development, starting with software development which includes system control software, the VxWorks embedded real time operating system (RTOS), communication between VxWorks and the NT system, graphic user interface (GUI) and diagnostic information available to the user through the GUI.

The key to flexibility for the UNAS' physical medium interface is the interchangeable mezzanine cards which are described next, followed by the universal network access engine (UNAE) which is the base card containing most of the protocol processing blocks. The special purpose mezzanine interfaces for video input and output are included here also.

The bit rate adaptive demux/mux (or BRAD) is described; and results of BRAD feasibility studies in CMOS and silicon-germanium technologies and subsequent development are covered, along with criteria for selection of a commercial chip with aspects of the BRAD functionality. Status of the BRAD development, which was not

completed due to program resource limitations are summarized.

The functional elements that make up the UNAE, the development and testing of which made up much of the work of this project, are described in detail. The blocks are generally assigned to handle one or more layers of the OSI protocol stack so that as the signal transits from the optical interface toward the back plane processing progresses from physical, to transport, to network layer and so on for any protocol. This is the how the description of each protocol design is organized.

The board level UNAE elements were designed to work with a high performance, compact-PCI platform. Rational for this choice and elements and performance capabilities of this platform are reviewed.

The focus of the UNAS effort was to create methodologies for processing different protocols without physically reconfiguring the system hardware. Development efforts for realizing, verifying and implementing hardware to process several protocols is covered. These include SONET, packet over SONET (POS), gigabit Ethernet, and two formats of uncompressed digital video commonly used in content creation and post production work. These serial digital video formats have been used extensively in DoD and hold promise when transport issues are resolved.

Also described is the development of an electrical interface and a custom protocol to allow the UNAS to act as a gateway between an OC-48 POS network and an all-optical, label switched router. IP packets from a POS network are extracted, the resulting packets encapsulated using HDLC, and custom labels are generated and passed to the all-optical switch and client interface developed by UC Davis.

Laboratory and field tests of the assembled functional blocks that comprise the Universal Network Access System are reviewed in section 4.3. Results of testing with commercial network and packet testers, routers and between UNAS-based nodes on wide area networks are included. These cover transport performance for difficult data such as SMPTE292 uncompressed HDTV and system and network performance such as jitter and dropped or reordered packets. Field testing included primarily packet over SONET testing, including the first demonstrations of a single 1.5 Gbps stream of video with embedded audio, transported across the US in a POS network with no dropped data over several hours.

Results of testing the label generation interface with the optical label switched router are also included.

Section 5.2 provides conclusions regarding system performance versus original expectations, suitability of architecture to meet the requirements, performance of design tools and methodologies to meet design challenges.

3 Methodology

3.1 Development of a System Requirements Document

Development of a system requirements specification started with creating of a User Requirements Document. The user requirements were based on the original proposal and the NGI UNAS Statement of Work for PR No. C-8-2586 dated 20 July, 1998; and are summarized below. The System Requirements Document was based on the following considerations and assumptions.

The Universal Network Access System (UNAS) will be comprised of a high speed protocol-independent front-end ASIC for clock and data recovery, followed by several FPGA's to do protocol-dependent framing, overhead extraction/insertion, payload extraction/insertion, error detection, and formatting for transmission across a common interface. System operation will be controlled by software which can provide system control and real time updates of network and traffic flow status.

The technology developed here will be compatible with rapidly deployable and reconfigurable networks, development and testing of new protocols, network monitoring and quality of service (QoS) testing.

The system will include the capability to provide synchronized time stamps sufficient to implement selected QoS monitoring and measurement capabilities.

The system will support inserting or accepting cells or packets directly to or from the main data path.

3.1.1 Platform

A platform will provide power and control for modular network interface and access cards. It may be either bench-top or rack mountable. A local or remote monitor and control interface will be provided. A hard drive and the ability to connect CD ROM drive and download CD-ROM data to hard drive will be provided. A standard network interface (such as Ethernet) is desirable and necessary for a remote user interface.

3.1.2 System Interfaces

The serial network interfaces implemented in the UNAS will comply with requirements of the selected protocols listed in Table 1, Table 1 UNAS Serial I/O Formats, with the appropriate interface board configuration and logic programming.

Each physical layer connection must provide the means to physically reconfigure the external serial connection to the system. This connection includes physical , ediua attachment connectors and conditioning elements specific to an interface port for an optical network or a source adhering to a video data format in Table 1. The port interface will support 150 Mbps - 3 Gbps.

3.1.3 Protocols

Only point-point, serial self-clocked, full duplex protocols transmitted at bit rates from 155 Mbps – 3 Gbps are to be accommodated. This excludes collision detection and handling, arbitrated loop and bus topologies. Only long-wavelength optical versions of serial protocols are to be implemented. The protocols are summarized in Table 1. No architectural decisions will be made that will disallow support of additional protocols and the system will retain the ability to support other current or future serial protocols. Standard specifications and RFCs will be used as guidelines for implementation.

Table 1 UNAS Serial I/O Formats

Protocol	Implement[1]	Support[2]	Comments
ATM over SONET OC-3c, OC-12c, OC-48c	•		
IP over SONET (PPP / HDLC) OC-3c, OC-12c, OC-48c	•		Packet over SONET
Gigabit Ethernet (IEEE 802.3ae)	•		Optical fiber transport
SMPTE 259 and SMPTE 292		•	Serial video/data on optical fiber or coax
Encapsulated data over SONET		•	No current application
ATM (unframed)		•	No current application
Future point to point protocols[3] compatible with architecture		•	Subject to available specifications and standards

[1]For protocols denoted for implementation, the current plan is to include the protocol as part of the UNAS program.

[2]UNAS architecture will allow processing of the protocols denoted for support through hardware and software adaptations.

[3]Extension to additional, future protocols requires that specifications are well defined, implementation is compatible with UNAE architecture, and protocol may be tested using available tools.

A copy of this table was submitted to the Government and approved with comments which were incorporated in the version shown. Based on discussions with Mari Maeda,

DARPA/ITO SMPTE 292 was implemented with the tradeoff that ATM over SONET would be supported, and implemented only if schedule and resources were available.

Electrical versions of certain data formats such as SMPTE 259 video may be accommodated by configuration of the physical layer connection.

WDM support can be provided only with external wavelength-selectable WDM access elements. In this case one WDM channel can be supported at a time by each UNAE card.

SONET features supported: The system will accept normal SONET input from OC-3 to OC-48 data rates, and will behave as operational equipment at the SONET level. Any SONET function implemented will be according to Bellcore GR-253 SONET specifications. Alarms, secondary channel, and policing functions will be implemented only as needed.

ATM features supported: Performs cell delineation and HEC correction, payload descrambling, and filters to drop/divert specified ATM cells. Although originally desired, the ATM design was not implemented. The architecture supports ATM.

PPP and IP support for Packet over SONET: HDLC and PPP processing will be implemented. Recommended practices will be followed for high bit rate implementations. PPP in HDLC framing over SONET will be supported per RFCs 1619 and 1662. PPP parameters will be negotiated as per RFC 1661 using LCP packets directed to the host processor.

Gigabit Ethernet features supported: Gigabit Ethernet following 802.3ae, full duplex optical transport over fiber, will be implemented.

Video features supported: If implemented, video data arriving in standard serial format such as SDI or SDTI will be segmented and packetized into ATM cells or IP packets for transport over SONET. SDI video received in ATM/SONET or IP/SONET will be reassembled and retimed for correct playout. Timing information may be recovered sufficiently to help diagnose errors in reassembling for display.

As the program unfolded some changes were made to Table 1. ATM on OC-48 SONET was planned as the first full protocol developed. Since this was one of the most computationally complex protocols planned, development and real time realization of this protocol was to constitute a feasibility evaluation for the proposed programmable network access approach (UNAE). However, a specific request from Mari Maeda, DARPA ITO, was to demonstrate transport of uncompressed SMPTE292 high definition video in IP packets over OC-48 SONET. Since this was considered as computationally difficult as the ATM transport approach, it was adopted as the feasibility test for the UNAE concept. Although a first implementation of the ATM test bench was complete, the ATM design was never completed. Nor were the multirate SONET at OC-3 and OC-12. Only OC-48 SONET (the most difficult by far) was implemented.

3.1.3.1 System Measurements and Status Indications

For each protocol to be implemented, parameters of interest for determining the state of

the link and of the network access point will be presented at a graphical user interface. These include parameters and indicators such as packets sent/received, Bytes received, frame check sequence errors, and CRC errors for IP packets; and loss of signal, loss of frame, and B1, B2 errors for SONET.

Future protocols: For each protocol supported by the UNAS, measurements will be displayed for that protocol, based on protocol definition and intended use.

3.1.3.2 Quality of Service

The system will be software reconfigurable so that quality of service (QoS) measurements can be done on selected protocols without reconfiguring system hardware, except to adapt to different physical layer connections at the serial network interface. The system will support filtering and extracting packets or cells for QoS measurements using filtering criteria, which may be stored or provided by an external source.

3.1.3.3 Additional Functional Cards

The architecture will allow the user to implement custom functions on separate circuit boards via the platform's bus system.

3.1.3.4 Network Ports

The UNAS will have a means to adapt to different network-specific fiber optic physical layer connectors in a modular manner.

The UNAS will receive and transmit serial data at bit rates in the range 150 Mbps – 3 Gbps.

3.1.3.5 Software Requirements

The UNAS software will allow the user to select the various parameters of a given signal. These include the physical layer, network protocol, transport layer, data encapsulation, and scrambling/no scrambling. Status information, such as signal loss, loss of frame, errors, alarms, etc., will be presented to the user through software.

The correct operating state may internally initialized or be established using communication over a user interface between the system host and remote host through Ethernet.

System operation will provide for protocol negotiation. For example, PPP will require session negotiation via messages for a point to point data link to be established. All message types must be tolerant of delays incurred by the UNAS processing.

It is not a requirement to maintain compatibility with software currently used in products at Tektronix.

The UNAS must power up as a 'stand alone' unit. Basic self diagnostics are required. Successful power up must be indicated on the unit. Similarly, the UNAS system must be shut down in a manner such that it will not cause network failure in any fashion.

The system may be programmed by user input to select a protocol for receive and transmit. The system architecture will support some degree of protocol discovery based on received data. Any UNAS reconfiguration must be accomplished in less than a second.

The UNAS system will allow user, field, or developmental software updates. This will include user interface and protocol processing algorithms. The method of updating this information is not restricted.

The UNAS will provide a method of externally accessing debugging information allowing verification of the system design and proper functioning.

3.1.3.6 User Interface

A User Interface is required to allow the user to setup and query the system for correct operation. There is no requirement to have a User Interface present on the UNAS unit itself. A Windows NT interface may reside on the host system to allow local unit setup. The operation of the User Interface will be dynamic as the UNAS is configured. Selection of a different protocol will require a new screen to appear and status information to be displayed. The User Interface status information will be updated approximately once per second. The User Interface may be implemented using more than one GUI, such as Windows for local and HTML for remote input, query and display. A remote User Interface, accessible through Ethernet is a desirable feature. The interface must be written using commonly available tools with the requirement that the browser on the remote device must be Netscape V4.0+ or Microsoft Internet Explorer V4.0+.

3.1.3.7 System requirements

Based on the user requirements and the resulting proposed architecture, a System Requirements Document was developed. That was followed by a UNAS System Architecture Document for hardware and software, which acted as the basis for most of the following system design work.

3.2 Architectural approach

3.2.1 System

The UNAS consists of one or more Universal Network Access Engine cards. Each card is identical and software reconfigurable to allow support of different network protocols. Figure 3 shows the block diagram of the UNAS, originally intended to use a PCI-based platform to provide power, display, user data entry, data interfaces, and access to the host processor.

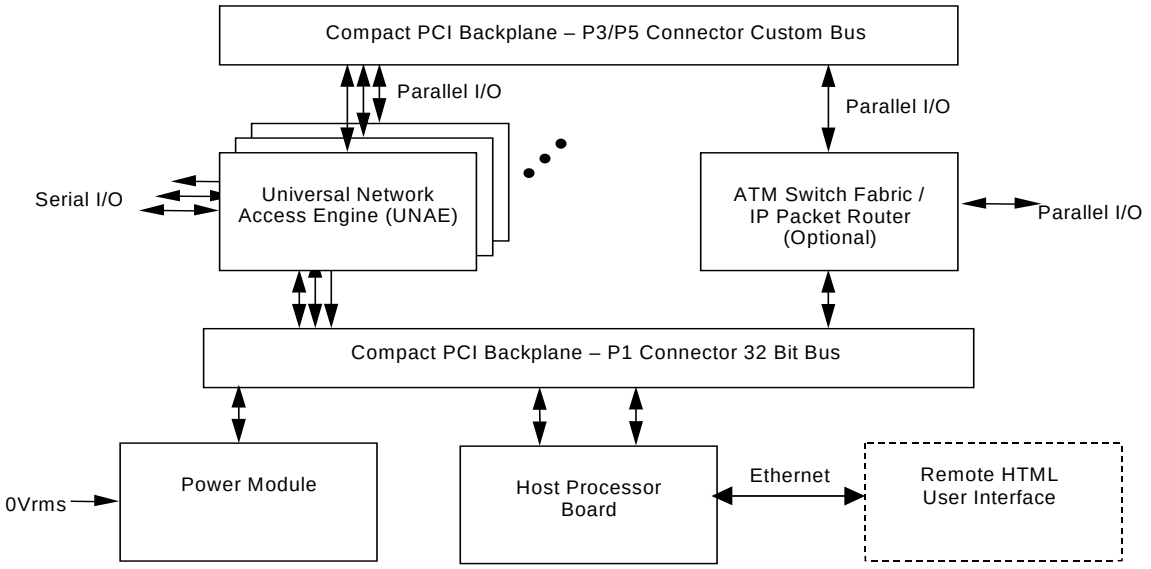


Figure 3 UNAS Architecture

The use of a 'compact PCI' backplane (PICMG 2.0 R2.1[i])with a proprietary, POS/PHY-3-like format was found to be superior to PCI and compatible with a co-developed platform at Tektronix. Use of this platform reduced development time and made the results compatible with Tektronix network test products. The core of the UNAS is the Universal Network Access Engine (UNAE), which supports individual serial I/O. The key element of the UNAE card is the physical layer protocol engine (PLPE), which can be re-configured to each of the supported protocols.

3.2.2 Card level Universal Network Access Engine

Each UNAE card is designed to be capable of receiving a serial signal, processing the protocol format, and passing the data over a parallel interface to the next card. The same card will also receive data from the parallel bus, encapsulate the data into the protocol, and transmit the signal through the serial interface. Transmit and receive protocols are required to match if duplex operation is demanded.

It is desirable for the software to provide an algorithm to detect the incoming protocol. It will then load appropriate FPGA code into the PLPE to do whatever processing is necessary for that particular serial format. The input data format is presented for display on the GUI. Upon receipt of a new serial input format, the software will re-configure the protocol.

The UNAE will have the capability of looping back data in both directions.

The UNAS team undertook an initial study of protocol characteristics looking for

similarities which could be leveraged in a protocol processing engine function. Protocols included in this study were ATM/SONET, IP in PPP framed in HDLC over SONET, Gigabit Ethernet, and cells from a burst switch such as that proposed by Turner[ii] at Washington University or Yoo[iii] at University of California, Davis. Common elements of these protocols led to the identification of several functional blocks which could be shared in processing of each protocol. These include clock and bit rate capture, frame boundary identification, scrambling, descrambling, cell header and payload processing and handling of cells or packets as the data format for communication between UNAE cards or bridge cards. The bit rate/clock capture and perhaps framing have enough in common to be captured in a single ASIC (also performing the high speed SERDES which requires the speed available only from an ASIC approach). This is especially true for OC-48 SONET framing which has significant processing overhead which must be done synchronously. The physical layer protocol engine (PLPE) will be best served by realization in a high speed electrically programmable logic device, also referred to as a field programmable gate array (FPGA) with a library of protocol processing images which can be rapidly loaded into the FPGA under software control. This choice of an FPGA to realize the PLPE provides a highly flexible architecture for investigating new protocol structures, and also influences the PLPE design approach. The functional element interfacing the cells and packets to the parallel bus (CPE or cell/packet engine in Figure 2) can be readily realized in an FPGA. If speed or reduced cost, are issues this can also be ultimately realized in an ASIC.

Burst data provides a special challenge for any architecture. In the UNAE bit rate identification and data capture may be done on burst data through the BRAD, but the UNAE cannot discover the protocol of data in bursts – it must be provided that information or data will be lost. The architecture also includes attached memory and first in, first out (FIFO) memories for rate matching across different clock domains.

Figure 4 shows the data path architecture for a UNAE and attached mezzanine board. The on-board i960 microprocessor on each UNAE was chosen to have 32-bit PCI interfaces to the Framer, PLPE, CPE and a specialized interface to the DDS and timing modules. VxWorks real time operating system (RTOS) runs on the microprocessor to handle real time functions, reading and processing information in FPGA and ASIC registers and passing messages to the host operating system as needed. The RTOS also handles link protocol negotiations as needed.

Additional useful functionality can be realized through the performance enhancement and flexibility from the on board processor. A reconfigurable network terminal device may be realized using the UNAE architecture. The additional ability to monitor individual network flows for quality of service (QoS) parameters will be of significant value. For a network through which a physical or virtual path is established, a network flow is a session of data transmission through an established network path. Data units establishing such a flow could be IP packets, ATM cells or data bytes depending on the network type. QoS characteristics of the data flow (network latency, latency variation or jitter and flow rate measurements in bytes/sec) will be important for characterizing both developing and

operating networks. A passive QoS measurement may be performed on packets by having the PLPE decode in real time the flow identification information contained in headers and diverting a subset of this "filtered" data to the RTOS for further processing, gathering of statistics, or similar functions Filtered and captured packets could to an extent be diverted to an additional processing target connected to the back plane or to the PCI bus. This functionality was not implemented in this study.

3.2.3 Mezzanine boards

The main UNAE board will have space for a single optical network interface mezzanine board that will provide electrical and optical connectivity appropriate to the chosen serial standard. Mezzanine boards may be changed to accommodate different physical connectors if necessary. The active physical layer connection can handle input and output connection simultaneously for full duplex connections. For special data formats such as SDI digital video, two smaller mezzanine cards will fit on a single UNAE. This is to accommodate existing video mezzanine boards from other programs.

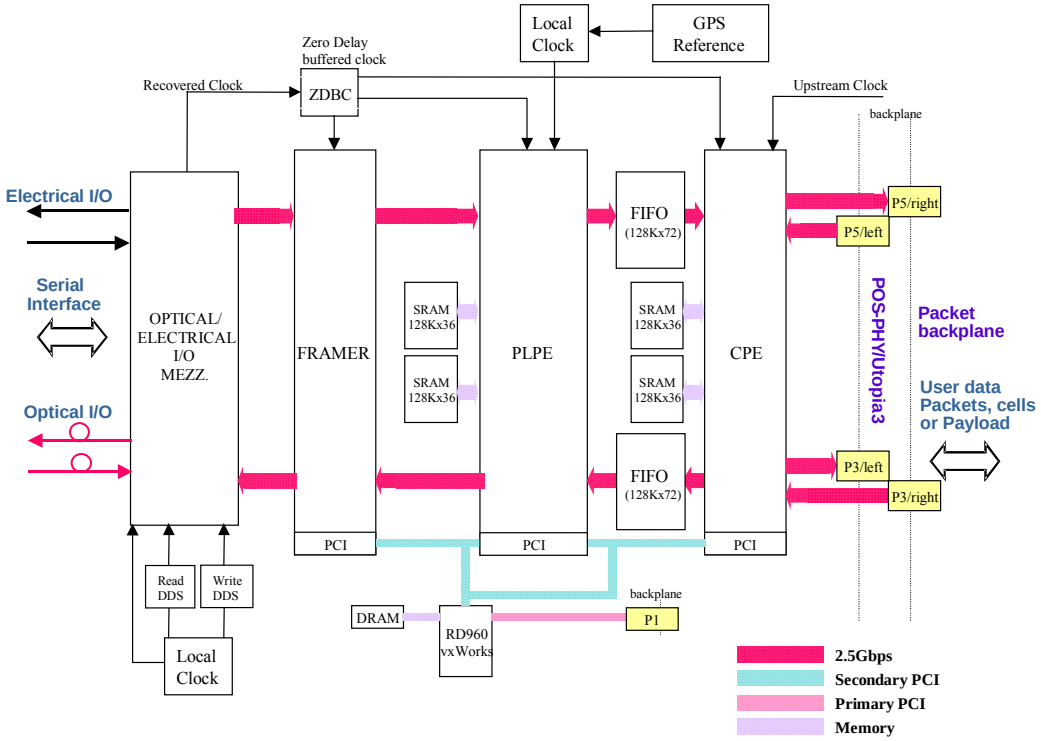


Figure 4 UNAE data path architecture

Clock and data recovery and Serialization/deserialization functions are performed on the mezzanine board. The BRAD ASIC or an equivalent functional block will accomplish this. A mezzanine may also be designed to hold commercial CDR and Serialization/deserialization IC's in the place of the BRAD. Details of the BRAD design

are included in section 4.2 below in this report. The serial data that enters the DeMux exits as parallel data over the mezzanine connector to the Framer.

3.2.4 Protocol processing engine

Once the received data exits the deserializer as parallel data, the protocol processing is relegated to the combined functions of the Framer, Physical Layer Protocol and Cell/Packet Engines (PLPE and CPE, respectively). Transmitted data also flows through all these blocks as indicated in Figure 4. The Framer, PLPE and CPE data processing blocks are each described individually in section 4 below in this report. The FIFO is a standard off the shelf component that is used for buffering between two clock domains when there cannot be assured perfect clock synchronization. The FIFO, in conjunction with PLPE and CPE functions may be used to assure data is removed from the FIFO at a rate that matches over a time window the rate of input to the FIFO. This is used among other things to assure correct play rate of video data.

3.2.5 System platform

A Platform Requirements document was created and released Aug 1999. A Compact PCI platform was chosen as it leverages ongoing work at Tektronix to provide a 2.5 Gbps backplane in this form factor. This will conveniently meet the needs of the UNAE POS/PHY-3 bus. Internally this platform is called Platinum and will be referred to by that name in this report.

By using this platform, development plans could include leveraging earlier work on a previous Tektronix developed high speed video network interface using a PCI platform with an i960 embedded processor running VxWorks. The Platinum platform used in UNAS is more compatible with telecom and network monitoring standard usage.

3.2.6 System Software architecture

A Software Requirements and Architecture document was created and released Aug. 1999. Windows NT 4.0 was chosen for the host platform. Linux was considered an alternative to NT for the host OS. However, NT was ultimately used because it has been supported and developed on Platinum for Tektronix products. VxWorks 5.2 on an Intel i960 microprocessor was chosen for the processing power on the UNAE card and was reused from another program to realize a system with reduced development cost and low risk.

The NT host local GUI connects to VxWorks through software passing messages across the PCI bus. A GUI communicating remotely for control and status using HTML or other technologies is required. It is a requirement that the GUI be dynamic in nature, having the ability to change and adapt to different protocols.

The Software Architecture is illustrated in Figure 5. There are several components designed for the NGI program, which span across three platforms.

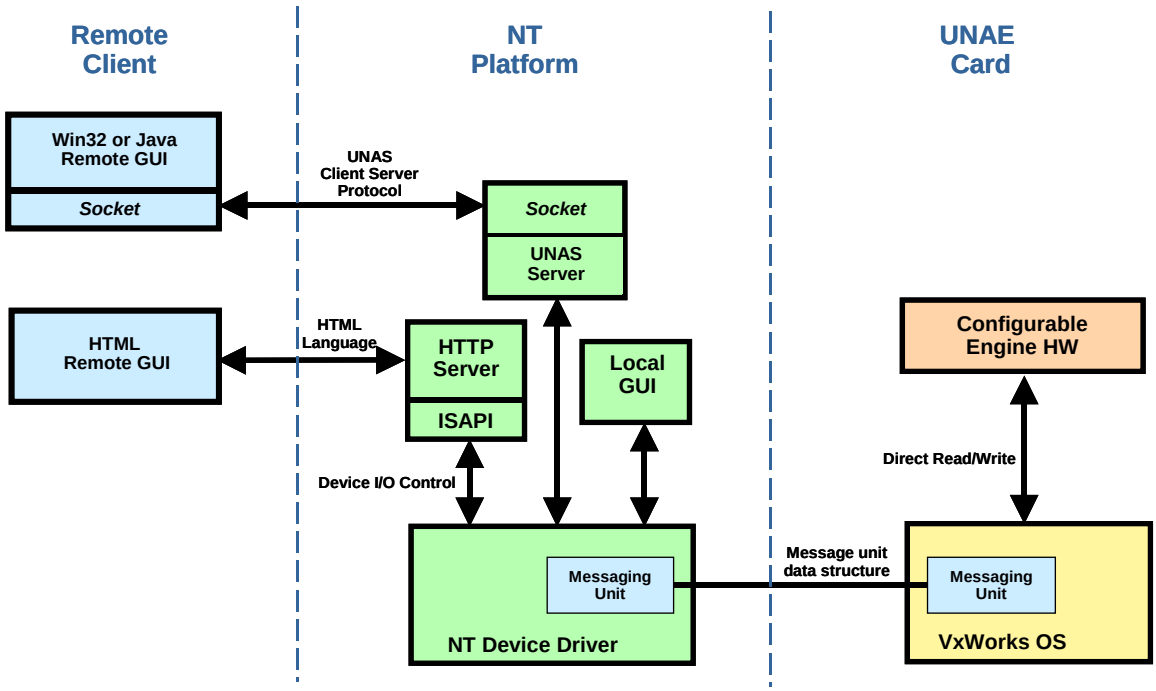


Figure 5 UNAS System Software Architecture

In Figure 5, there are several blocks designed for the NGI program to support the UNAS system. Each of these will be described in following sections.

VxWorks OS - The largest software block is the VxWorks Operating System with the VxWorks Application Module. The VxWorks Application Module (not shown as separate from the OS) contains all the FPGA images, protocol negotiation software, and UNAE specific drivers, and messaging for the GUI. The Application Module is loaded through the NT driver and dynamically linked to the VxWorks core OS.

NT Device Driver – Scans the PCI bus for UNAE cards, initializes them, and loads the Application Module. Once the module is connected, the Device Driver acts as a conduit for messages between VxWorks and the GUI.

Local GUI – The Graphical User Interface designed for the UNAS system for control and status of each UNAE card in the system. Dynamic in nature, it adapts to the various protocols supported in the NGI program. Written using Visual Studio, the GUI is designed for ease of use and the look and feel with other Windows applications.

HTTP Server – Uses the ISAPI interface to connect with a HTML interface for control and status information. Server translates HTML messages into DeviceIOControl messages.

UNAS Server – Uses sockets for connecting to a JAVA or Win32 application. Server translates proprietary messages into DeviceIOControl messages.

HTML Remote GUI – A UNAS web based application for control and status of the UNAS system from a remote location.

JAVA or Win32 Remote GUI – A UNAS application running on a remote host for control and status of the UNAS system. Connects to the UNAS Server.

The VxWorks software also contains data structures for maintaining state information and status information. The PPP Negotiation and Gigabit Ethernet Auto Negotiation control software and state machines are also located in the VxWorks module.

3.3 Design tools and hardware selection methodology

The Universal Network Access System (UNAS) was conceived as a high speed front-end for serial clock and data recovery, followed by several FPGAs to do protocol-dependent framing, overhead extraction/insertion, payload extraction/insertion, error detection, and formatting for transmission across a UTOPIA/POS interface. These functional blocks required high performance, sophisticated designs. This provided both the need and the opportunity to select tools that would provide the most powerful design environment for manipulating data formatted in cell, packet and transport frame protocols.

3.3.1 Hardware design and simulation requirements

The UNAS requirements to implement several high speed communication protocols make the hardware design a difficult task. It was initially thought that a complete system model would be needed, simultaneously describing the ASIC's, programmable logic, processor, and control software, in order to simulate operation of the entire system for the various supported communications protocols. A survey was performed to identify software products which are primarily designed to model the entire electronic system comprising multiple chips, programmable hardware and software. Results of this survey, taken in 1999, are summarized here.

Opnet Mil3 is a high-level modeling environment meant to simulate the performance of networks (switching delays, queuing models, etc.). According to available company information, the software has the ability to model every cell in an ATM stream, and can also model call set-up. Models are written in a graphical environment; while behaviors of elements are written in C. Prepared models are available for PPP, gigabit Ethernet and ATM.

Object GEODE from CS Verilog uses graphical input (message sequence charts) to generate an SDL model of a system. SDL is a Specification and Description Language standardized as International Telecommunication Union - Recommendation Z.100 and is designed to communicate extended finite state machines expressed as processes. The system is simulated by executing SDL code. C code can be generated from SDL for supported processors. At the time of the survey there was no way to get to hardware designs directly from the SDL model although a company named Arexsys was working on a solution that was a minimum of 6 months out.

Cadence BONeS Designer is a Block-Oriented Network Simulator for the design and

analysis of system architectures, networks, and protocols. It consists of a Data Structure Editor, a Design Simulation Manager, a Finite State Machine Editor, an Interactive Simulation Manager, and a Project Editor. It has the capability of incorporating SDL code from outside sources into its simulations, but no facility to generate hardware designs. Models available of interest include ATM and Ethernet.

Telelogic has developed a suite of tools for modeling and simulating systems using the SDL language. Tools are the Object Model Editor, the MSC Editor, the SDL Editor, the SDT Analyzer, the SDT Simulator, and the Cadvanced/Cmicro Code Generators. The tools are reported to be good for state machine design and poor for algorithm design. There is no path to hardware from the SDL models: SDL code can be imported to Cadence BONeS for simulation, but there is no autogeneration of VHDL/Verilog. ATM models were pending availability from second party supplier Cellware.

With this and similar information in hand, starting with a high level system model with the intent to drive it down through successive steps finally to a hardware description language (HDL-level) hardware design was seen as a roundabout and difficult method to achieve success. A high level system model was then viewed by the team as a process which would draw resources away from the already largely specified design task at hand and would not provide a more automated or more efficient design or testing environment nor would it contribute substantially to enhancing the probability of success of the UNAS design. There are still three ways a model or design environment can help realize a successful design: 1) It can generate the architecture, 2) it can verify the architecture, and 3) it can verify the design implementation. Of these, benefits 2) and 3) were deemed to be mostly directly beneficial and drove our selection of the type and source of EDA tools.

In order to implement protocol-specific hardware as loadable images in FPGA's, several different designs must be quickly implemented in an HDL that could be synthesized to FPGA hardware. It was desired that the hardware design tools reduce the design effort required to implement multiple protocols, and make it possible to easily reuse design blocks and to allow easy modifications to designs.

The following were requirements for hardware design/simulation tools:

1. Generate Verilog HDL code for FPGA synthesis
2. Provide a design framework that reduces design time for network protocols
3. Provide for reuse of design blocks and ability to make design changes easily
4. Simulate the design quickly (OC-48 framing requires processing $>10^6$ data bits)
5. Provide a capability for including the hardware design into a system model if needed

Software from six vendors was evaluated for use in the NGI Universal Network Access System (UNAS) design. The design software was evaluated with respect to the criteria listed below:

1. Appropriateness of the tool for the design problems to be undertaken
2. Efficiency of designing with the tool compared to coding directly in Verilog HDL

3. Usefulness of tool output code (including or in addition to Verilog output)
4. Ability to link hand-coded Verilog into design
5. Ease of learning the tool and becoming productive
6. Ease of design changes and the ability to reuse design blocks
7. Availability of timely support
8. Ease of project documentation

The Cadence software has been installed as the tool of choice for Verilog simulation at Tektronix (Verilog-XL and NC-Verilog). Since other tools did not provide substantial additional benefits, the Cadence tools were used for simulation. Simulation times using Verilog XL were estimated in advance to be up to several hours for large FPGA's using the fastest Solaris platform currently available.

3.3.2 Hardware synthesis requirements

Verilog HDL was the design language used for the UNAS project so a synthesis tool that takes Verilog netlists and generates FPGA code was needed. The tool must generate code specific to the type of FPGA chosen for the design (for example Altera Apex or Xilinx Virtex).

The following are requirements for hardware synthesis tools:

1. Good coupling to the design tool chosen
2. Generates fast and efficient code for the hardware device chosen
3. Synthesis parameters settable for parts of a design to tweak performance
4. Easy to debug designs

Synplicity Synplify software is installed as the tool of choice at Tektronix for synthesizing Verilog designs for FPGAs. It meets all the requirement s above. Since other tools do not provide substantial additional benefits, Synplicity software was used.

3.3.3 Hardware verification requirements

An independent check for correctness of implemented communications protocols in the UNAS hardware and software designs was needed. Consequently, one or more engineers different from the hardware design team assembled a protocol test bench for some of the protocols implemented. The test benches consist of a packet/stream generator and a data checker to verify correct handling of the data. The test benches were designed using a commercial software tool.

The following requirements were identified for hardware verification tools:

1) Ability to generate a complete series of test frames to exercise the frame-synchronization circuits at the UNAS input (several million bits for OC-48 SONET frames)

2) Able to model the hardware at a high level of abstraction
3) Generate reuseable code for test benches
4) Increase efficiency in coding stimulus generation and data-checking over that possible using C or Verilog
5) Provide a high design coverage for verification
6) Must be compatible with other tools chosen for the design

3.3.4 Tool selection

Evaluations of tools for hardware design, simulation and verification included C-Level Design from C2Verilog, ArchGen/ASVP Lab from CAE-Plus, Protocol Compiler from Synopsys, Cierto Hardware Design System/Signal Processing Workshop from Cadence, and Renoir from Mentor Graphics. Also evaluated were design verification tools suites, including VERA from Synopsys and Design VERIFYer/Design INSIGHT from Chrysalis. The following software was chosen for the design and simulation of the NGI PLPE hardware:

Design: Synopsys Protocol Compiler, Verilog HDL

Simulation: Cadence Verilog XL, NC Verilog

Synthesis: Synplicity Synplify

Verification: Synopsys VERA

Protocol Compiler is specialized for designing hardware systems where the input/output is data and control information contained in frames. It is specifically intended for the creation of state machines for the control logic of communication protocols. Through a graphical interface the designer associates actions with specific parts of data frames. The software then synthesizes protocol control logic and generates HDL code for the design. Protocol Compiler's design also facilitates making changes to the design: changes are made at a high level in Protocol Compiler and new Verilog code is generated by the tool, which can then be simulated and synthesized into hardware. There is some issue with the speed of simulation of Verilog code where simulation of framing mechanisms of different protocols are involved. Cadence Verilog XL was adopted for HDL simulations. Verilog HDL allows linking of hand-coded modules into Verilog design output of Protocol Compiler. It is a tool that has been in use at Tektronix and is supported. A large amount of data will need to be processed to test framing. A compiled Verilog simulator could be used to bring simulation times down to a reasonable duration.

Synplify was chosen for synthesis of the Verilog design to hardware. It was the consensus of the design group at Tektronix that this tool currently provides the best FPGA implementation of a hardware design from Verilog code. Experiments with synthesis of Verilog code for Altera PLD designs indicated that it compares well to designs synthesized with Altera's dedicated software.

Three hardware design engineers received training available for Protocol Compiler before commencing design work.

For hardware verification of Protocol Compiler designs, Synopsys VERA software was chosen. This software provides significant advantages over writing test benches in either Verilog or C code in terms of both efficiency of code and effort involved in developing it. VERA is a superset of Verilog code that is enhanced for the purpose of test bench generation. Some weight was given to the fact that the software is from the same vendor as the design software that will be used, putting the responsibility for software compatibility with the vendor. Efforts have also been underway by Synopsys to make the VERA verification language a standard.

3.3.5 Hardware assumptions and selection

Following an initial study of protocol similarities which could be leveraged for a protocol processing engine, common elements of these protocols led to identification of several functional blocks which could be shared in processing of each protocol. These include clock and bit rate capture, frame boundary identification, scrambling, descrambling, cell header and payload processing and handling of cells or packets as the data format for communication between UNAE cards or bridge cards. Due to the extremely fast real time processing needed for the serial front end (clock and data recovery and serialization/deserialization) an ASIC was chosen. The BRAD was chosen to fulfill this role as described. The framing function is also computationally demanding even though processing is done on demultiplexed data. This is especially true for OC-48 SONET framing which has significant processing overhead which must be done synchronously. There is some risk in assigning an FPGA to this task but the flexibility suggested this is the best way and was worth the effort. The physical layer protocol engine (PLPE) will be best served by realization in a high speed electrically programmable FPGA logic device with a library of protocol processing images, which can be rapidly loaded under software control. This choice of an FPGA to realize the PLPE provides a far more flexible architecture for investigating new protocol structures, and also influences the PLPE design approach The CPE which interfaces the cells and packets to the parallel bus can be readily realized in an FPGA. The architecture will also include attached memory and first in, first out (FIFO) memories for rate matching across different clock domains and embedded SDRAM for buffering data for in line calculations, reordering of data streams and similar manipulations.

The BRAD ASIC was defined to be an internally designed CMOS chip that would handle the clock and data recovery (including burst data) and mux/demux functions. A feasibility study was undertaken to assure that it would perform as needed at the highest data rate for the most demanding protocol. Based on the results of the CMOS feasibility study an alternative of using a SiGe process instead was proposed. This technology was shown to be extendable to 10Gbps.

Once all major functional blocks and the overall architecture were determined, with I/O dedicated to a mezzanine that includes the BRAD and a set of duplex electrical and

optical interfaces, decisions were made on the nature of, generation, and distribution of all clocks. Three independent direct digital synthesizers (DDS's), with timing references optionally derived from GPS satellite UTC, provide reference clocks for the I/O mezzanine receiver and transmitter, and the card to card back plane data interface. The I/O mezzanine provides synchronous interface timing for transmit and receive paths and to the remaining blocks distributed on the UNAE motherboard. A high density FPGA family (Altera APEX, 400K-1.5M gate) was tentatively chosen to realize the implementation of the Framer, PLPE, and CPE blocks. The main circuit board and attached mezzanine is to fit into one slot of the industry-standard compact PCI (CPCI) chassis. Although Altera and Xilinx FPGA components are comparable offerings, the Altera toolset was considered easier to use. This influenced selection of Altera's APEX FPGA family.

Initial plans were to leverage earlier Tektronix work, using a PCI platform with an i960 embedded processor running VxWorks. This was connected to an NT host through a system of software passing messages back and forth across the PCI bus. However, based on several considerations a Compact PCI (CPCI) platform was adopted instead. There was ongoing work at Tektronix to provide a 2.5 Gbps backplane in this form factor, which will conveniently meet the needs of the UNAE back plane bus. The i960-RD processor, VxWorks and NT host remained. The on-board i960 microprocessor on each UNAE was chosen to have 32-bit PCI interfaces to the Framer, PLPE, CPE and a specialized interface to the DDS and timing modules.

3.3.6 Design tool and manufacturing support

Tools support was provided for the software design tools by Tektronix' Electronic Design Tools group, which is part of Central Engineering services. These services also include board layout support and mechanical engineering support. This included SW tools maintenance, license check out and server resources. All technical work on the tools was done by the NGI project team. Only the Synopsys Protocol Compiler and VERA were not Tektronix standardly supported tools.

Tektronix has an infrastructure for new product development and introduction that includes component procurement, fabricated board procurement, automated board manufacture and assembly, and custom hand-placement for special parts that are not compatible with volume tooling. This project used these faclities to complete builds of working boards in a timely manner. Reworks for incorrect and damaged parts were also supported by the new product manufacturing facilities.

3.3.7 Functional and performance testing

3.3.7.1 Simulation and test benches

Designs, once realized were simulated at several levels: First, using functional performance test benches such as VERA or Verilog to make sure the logic is implemented correctly. Protocol Compiler includes a simulation tool that does Verilog simulation. It also includes a waveform editor to simulate the logic timing of the design at

a high level. Thirdly, pre and post-place and route timing checking: Quartus, the Altera floor planning tool, has timing simulators as does Synplify from Synplicity.

Developing and using a test bench can be complemented by other means for reduced risk. For example the Framer and PLPE elements were intended to be advanced programmable logic devices, each with a library of protocol-specific images to be loaded and executed. One critical task became designing the executable image for each chip for each protocol. Realization of each protocol relies on use of a published standard. Misinterpretation of the standard can lead to subtle errors which may be difficult to debug. A verification methodology was chosen to address this issue. For each protocol, a hardware design based on the written standard was done by a knowledgeable engineer. A second engineer independently developed a test bench based on the same standard. Verification of the hardware design using the independent test bench provided confidence that both implementations were correct. Three software design engineers received training for the VERA verification language before commencing design verification work.

Using SONET as an example: A SONET Framer test bench design was developed in the VERA development environment. SONET frame recognition, pointer processing, and scrambling/descrambling portions were included. Passing the test bench made the SONET receiver more reliable, working gracefully under erroneous conditions, e.g., when bit errors are injected. VERA will also be used to make the SONET transmitter more configurable.

The next step in the hardware design task is to synthesize the FPGA. This is where the feasibility of the approach becomes more clearly defined. Protocol Compiler was proven to be an efficient design entry and verification tool. The Verilog files produced by Protocol Compiler were input to Synplicity to produce FPGA images. Timing checks will be performed on the Synplicity output to verify that the FPGA is fast enough. Once these timing checks have been made an evaluation of the design approach and its limitations may be made.

The PLPE OC-48 Packet-over-SONET implementation including extensive testing of the PPP receive and transmit designs were done using VERA test benches, and error free operation was observed

Alternatively, several modules were designed using a more conventional approach, either because the design was simple enough for this to be more efficient (Cell Packet Engine block) or because it was imperative to have more direct control over the resultant HDL code for enhancing the performance. The Gigabit Ethernet design was simulated using a test bench created using Verilog. There were four stages of simulation and verification performed during the design and debug of this system. First individual portions of each FPGA in the system were simulated with a functional RTL test bench designed to verify that specific portion of the design. Second, an RTL simulation was performed on each FPGA in the system. Third, the entire system was functionally simulated using a top-level test bench. Finally the entire Gigabit Ethernet system design consisting of three FPGA’s was simulated using a Verilog model for each FPGA which included back-annotated timing information.

3.3.7.2 Laboratory testing of functionality

Design tools were chosen to provide as accurate and thorough a simulation of designs and components as practically necessary. However, verifying performance in the lab is necessary to successfully debug designs, verify successful system level integration and determine performance limits before attempting any network testing. Efforts included improving our testing infrastructure and exploring how close a high performance (gigabit and 2.5Gpbs) network could be brought to the Tektronix site for extended testing.

To meet the requirement of realizing a UNAS evaluation and demonstration environment using primarily lab-level testing, capital purchases of packet over SONET and gigabit Ethernet signal generation and measurement equipment were made by Tektronix in order to support program activities. Network and packet testers were purchased from both IXIA and Spirent/Adtech. A GSR12000 router with multirate OC-3/12/48 Packet over SONET and gigabit Ethernet line cards was purchased from Cisco for an internal project; and this was also made available for the current NGI program. Measurements on early versions of a UNAS system, when a functional BRAD would not yet be available would be needed for program progress. This was to be addressed by developing hardware and software for a dedicated ASIC test & operation evaluation environment, consisting of an operations platform, a test protocol generation card, and an ASIC test and evaluation card, if needed. An operating system, bus interfaces and driver software for the test environment would all be required for testing.

The latter challenge was addressed using the UNAS itself as a development platform. It was able to generate, in conjunction with external test equipment, all test signals required to test system component functions as development progressed.

3.3.7.3 Gaining access to field testing on networks

Once laboratory experiments showed feasibility and provided performance parameters, it made sense to broaden the scope to connecting into actual operating networks to exercise UNAS capabilities. Early and continuing efforts addressed collaboration and cooperation with other NGI members on the planning and conduct of test bed demonstrations to address hardware and software interoperability issues.

Several activities have occurred throughout the program to develop collaboration opportunities and procure access to research test bed networks:

Tektronix initiated discussions with Ben Peek of GST Telecom and Hal Edwards of Nortel regarding interoperability of the UNAS technology and NTON II. Discussions were held with Ben Peek, GST starting August 20, of 1999. Considerations included video transport, data security and network monitoring functions. Additional conversations were held with Bill Lennon, responsible for NTON applications, regarding opportunities for collaborating with NTON II. Meetings with Ben Peek also addressed the prospect of fielding both passive and active IP flow monitors on NTON II. And an architecture for a passive monitor (based on the UNAS) was discussed. This was to

considered a promising technology with far reaching value to a research network like NTON II and with significant potential for commercial use. Follow up meetings were held October 8, November 18 and December 3. From January – June 2000 discussions started with GST, Avici and Tektronix to establish a colocated development lab, using floor space donated by Tektronix, for advanced development, interoperability experiments and Beta testing of new commercial equipment. This co-locate space was to be a key element of substantial value in developing the demonstrations in the UNAS program. However, by June GST's financial situation required resolution under Chapter 11. Time Warner Telecom purchased most of the assets of GST; but NTON II and the co-located development lab were unresolved issues.

In September, 1999 UNAS Program Manager Kirk Boyer contacted NGI Principal Investigator Chandrasekar Chandra at CSU to discuss video transport needs of his program. It wasn't clear that the CHILL program had a clear view of needed collaboration at that time. Chandra was invited to contact Tektronix for further discussions. These discussions, primarily at subsequent NGI Principal Investigators' meetings, indicated that the needs of CHILL facility were not aligned with UNAS capabilities.

On October 13, 1999 Kirk Boyer visited Traci Monk and NGI Principal Investigator K.C. Claffy of CAIDA to discuss respective plans and potential for collaboration. An NDA was submitted to CAIDA by Tektronix on October 19, which was not processed by CAIDA due to conflicting intellectual property issues with another corporate sponsor. One area of mutual interest was the ability to optically filter DWDM data to select flows belonging to a specific lambda and subsequently filter the IP or ATM flow for QoS applications. Because this second attempt to collaborate on QoS measurements faltered, priorities in the program were redirected to solving other significant technical challenges. One of these was transport of very high data rate, uncompressed HDTV video over IP networks.

Phone conversations were initiated by Tektronix in late 1999 with NGI Principal Investigator Alison Mankin of ISI East to explore interest in transport of both uncompressed and compressed HD video and audio data. Collaboration topics included helping to specify HD acquisition and display equipment and to contribute to a proposed Internet draft to IETF: "RTP Payload Format for Uncompressed HDTV Video Streams." This included making sure the proposed payload format can accommodate the SMPTE 292M serial transport steam format to be used in the UNAS program. The proposed transport format is RTP/UDP/IP with special RTP header information to assure compatibility with other sources and receivers of the video. An IETF draft (draft-ietf-avt-smpte292-video-00.txt, July 13, 2000) was prepared by Allison Mankin and Ladan Gharai at ISI East, David Richardson of the University of Washington and Tektronix. A requirements document for video operation of the UNAE board was also written (NGI UNAS Video Requirements Document, revision 1.0, July 31, 2000). A demonstration of HDTV transport using RTP/IP in Packet over SONET using Internet2 and the Supernet was planned for November at the SC2000 meeting in Dallas, TX. The UNAS hardware was unable to meet necessary internal timing requirements and the intended

demonstration was not shown at the SC2000 meeting However it did go on to make a significant, positive impact at SC2001 in Denver, CO in the NCO ITR&D booth. With our partners ISIE, U of W, Internet2 and Level3 the UNAS transported 1.5Gbps HDTV losslessly in IP packets through an OC-48 packet over SONET network from Seattle to Denver.

Efforts to place a GST/Avici/Tektronix collocated facility on Tektronix' campus failed. However, use of dark fiber on Tektronix' campus was considered, in conjunction with DWDM equipment from LuxN, to field a demo of the project's results on a local area network. Discussions were initiated with ATT Broadband to provide managed OC-48 service and with MFN to provide dark fiber, originating on the Tektronix LAN, to a carrier hotel in Portland. A carrier at that POP could carry the data to a Seattle GigaPOP for connection to I2/NTON and HSCC for a WAN demonstration. This would be an alternative to transporting project equipment to University of Washington to act as the western terminus of demonstrations. Until recently, there was no lit up OC-48 link from Portland to Seattle. Additional efforts with the Portland Exchange, a consortium of network users developing a network centered on the Pittock Building in Portland and with Tyco Communications have not yielded access to 2.5Gbps links without restrictive costs.

Finally, based on communications with NGI Principal Investigator Ben Yoo, UC Davis, an additional task was added to the UNAS program to enhance the UNAS engine to support Optical Label Switched Router technology through a collaborative Subcontract. The work was funded and has resulted in demonstrating successfully a gateway between a conventional packet based network (OC-48 packet over SONET or GbEthernet) and an all optical packet switched network running at 2.5Gbps.

Results of testing the UNAS on wide area networks are covered in section 4.3 of this report.

4 System Development

4.1 SW development

4.1.1 Design objectives

The software allows the UNAS system to power up as a stand alone system, allowing the user to select various protocols and operating modes and to monitor performance of the network link with responses on the order of seconds or fractions of seconds. The operating system is an OS in standard use with ready access to modifications and upgrades. It must run diagnostics to assure full boot is achieved. The UNAS acts as an attached network element so that software on the protocol processing boards must be able to interact with and process the data flow fast enough that normal operation of the network and attached network elements are not impeded when communicating with or through the UNAS. Link negotiation with other network elements needs to be supported. Turning on and off the system must not interfere with other network elements in any way except to announce and acknowledge intent to disconnect active links. The software must support selection and monitoring of various parameters of the signal such as network protocol, physical layer, transport layer, data encapsulation and scrambling. Status such as alarms, errors, and loss of signal must be made available to the user or system controller on demand. Exchange of information between the user and system needs to take place through a simple to use and informative User Interface.

Software, firmware and user interface modifications and upgrades should be realizable through a network interface such as Ethernet.

4.1.2 Summary of Work: Software design and implementation

All portions of the software described in this section have been architected, designed, and implemented for the Next Generation Internet (NGI) program, with the exception of the two operating systems purchased from external vendors.

4.1.2.1 Operating Systems

The operating systems for the UNAS system consists of a host operating system (OS) for user interaction and an operating system for real time signal processing. The two operating systems communicate with each other through a predetermined suite of messages. These messages allow the transfer of small amounts (less than 512 bytes) of data between them. These messages allow setup information, status information, and conditional information to be passed between the host OS and the UNAE OS. It is not the intent of this project to pass the raw high-speed data stream from a selected protocol to the host processor by any method.

Host Operating System

Windows NT 4.0 with Service Pack 5 is used on the host processor for the operating

system on the Compact PCI platform. The User Interface is constructed using Visual C++. Windows NT was chosen for common knowledge of use, system updates, and User Interface development tools. The development platform is a PC workstation.

VxWorks

VxWorks version 5.2 real time operating system is used on each individual UNAE card. VxWorks is a proven operating system used by several groups within Tektronix. Each card provides an interrupt to the on board local processor every 100 ms. The interrupt handler gathers protocol, network, etc. status information and keeps this status information locally until it is requested by the host system. The VxWorks operating system runs on an Intel I960 microprocessor located on each UNAE card. This model was selected to maintain all processing of the protocols on the board level rather than the host microprocessor. The development platform is a UNIX workstation running Solaris.

4.1.2.2 System Operation

The UNAS software allows the user to select the various parameters of a given signal. These include the physical layer, network protocol, transport layer, data encapsulation, and scrambling. Status information, such as signal loss, loss of frame, errors, alarms, etc., is presented to the user. The status information is gathered from the FPGA registers and kept in data structures in VxWorks. The system operation is described in the following sections.

System Boot

On initial power on conditions, the Compact-PCI bus resets the individual UNAE cards and then proceeds to boot the host operating system. Initialization of host hardware, file systems, and Windows User Interface is also included.

UNAE Boot

The UNAE cards, upon receiving a reset on the PCI bus, resets the on board processor, start the VxWorks operating system software located in Flash ROM on the card. This VxWorks image is the basic core operating system and does not have any specific hardware drivers for any of the protocols or hardware. The VxWorks image is limited to the I960 processor, PCI bus, FLASH ROM, and dynamic RAM.

The VxWorks core operating system image is compressed in the FLASH ROM. At reset, the I960 processor will start executing the image from FLASH ROM. During this process, the I960 will configure itself from the initialization portion of the image. The processor will then uncompress the VxWorks image and place it in dynamic RAM. The VxWorks operating system is then started.

The basic VxWorks operating system is up and running. It has a basic message system and is waiting for a message to be received. The message it is waiting for is from the Windows NT driver with specifics for the VxWorks application to be loaded into dynamic RAM and then dynamically linked to the core VxWorks operating system.

Total time to this point from power applied is on the order of a few seconds.

Windows NT Boot

The Windows operating system is also being brought up when power is applied. This is a standard release of Windows NT 4.0 with Service Pack 5. This will take several minutes.

Starting the NT Driver

At the point that the Windows NT is up, then the NT Device Driver can be started. The NT Device Driver was designed with several functions. The NT Device Driver will first scan all PCI slots in the mainframe looking for UNAS cards. Once it finds a card, it will initialize the card. The NT driver will then allocate contiguous memory and load a VxWorks Application Module from the NT harddisk into the NT controller's allocated memory. The NT driver will then send a message to the VxWorks on the UNAE card with the NT memory information of the VxWorks Application Module. The core VxWorks operating system on the UNAE card will receive the message and set up a DMA transfer based upon this message. The DMA transfer is performed by the I960 on the UNAE card. After completion of the transfer, the VxWorks application module is then dynamically linked to the VxWorks core operating system.

The NT Device Driver itself is "unas.sys" and is located at C:\WINNT\system32\drivers on the host platform.

The NT Device Driver fetches the VxWorks Application Module from a specific location on the hard disk. The name of the file is "unas_drv.o" and is located at C:\WINNT\system32\drivers on the host platform.

VxWorks Application Module

This module is the largest piece of the software system developed by the NGI team on the UNAS. The module initializes all of the hardware, loads the FPGA images (the FPGA images are embedded in this module), installs a new message system, destroys the old messaging system, installs interrupt handlers, contains all status and state information, contains routines for PPP negotiation, Auto Negotiation for Gigabit Ethernet, FIFO level managing routines, and more.

The VxWorks application module is compiled on a UNIX platform using GNU tools. The target processor is the Intel I960 microprocessor.

Messaging System

The messaging system is the method of communication between the NT controller and the VxWorks operating system. Messages are designed on this program to provide setup, control, and status information. The VxWorks module is loaded by the VxWorks core operating system by a DMA transfer. Once the transfer has completed, the VxWorks OS will first dynamically link the new module. The initial messaging system is then removed and a new messaging system for the UNAE is installed. The initial messaging

system consists of four basic commands to set up the DMA transfer. The new messaging system contains close to thirty messages specific to the UNAE and the protocols supported.

The messaging system is implemented by using the internal mailbox registers in the I960 processor on the UNAE board. The NT controller communicates with the VxWorks through this path. The NT controller will place a message in the mailbox registers and then the I960 processor will then receive an interrupt that a message has been received. The VxWorks will then process the message for the setup or status requested.

The messages are in the form of basic commands. Example are: MU_GET_STATUS, MU_SET_PARAMS, MU_RESET_HISTORY, etc. All state information for the protocols are in the VxWorks portion of the software and are supplied to the NT controller by request. There are approximately thirty messages for each UNAE card.

Each message has a distinct structure shared by the NT driver and VxWorks. For example, the MU_RESET_HISTORY message is a command to reset the "history" portion of the status and counters. It requires no other parameters. The VxWorks will then return an acknowledgement to the NT controller when the task is complete. A MU_GBIT_SET_PARAMS message on the other hand will set the 48-bit source MAC address, a 48-bit destination MAC address, the transmit clock source, and diagnostic loopback type, all in the same message.

The messages are aligned in both the NT driver and VxWorks by defining the messages, values, data structure, transfer of messages, etc in two files. They are "deviceMsgs.h" and "deviceMsgs.c". These two files are used to compile the NT Device Driver, the VxWorks Application Module, and the User Interface code. Once the cards are initialized and the VxWorks Application Module loaded, the NT Device Driver is basically nothing more than a conduit of these messages from the User Interface to VxWorks. Care was taken in the program to ensure the messages stayed in sync throughout the entire software design.

There is also a window on the Windows NT desktop that can be started. It is the VxWorks "shell". This allows the user or engineer to get access to the VxWorks shell commands directly. The method used for gaining access is through a second set of I960 mailbox registers that are specifically reserved for passing of VxWorks commands. The "shell", similar to a UNIX command line, is used to gain access directly to registers, memory, and functions in the VxWorks software.

FPGA Images

The FPGA images are compiled, by the hardware team, into "ttf" files . These are ASCII text files corresponding to the binary value to be programmed by the FPGA loader routine. These files are constructed by the software build environment with headers and trailers on each file, making them into "C" language arrays. These arrays are then compiled into the VxWorks Application Module.

The routine to program the FPGA is also in the VxWorks Application Module. It will

take the array values and then program the FPGAs. It will also report the status of the load process through the VxWorks shell.

UNAE Hardware Initialization

The application module then loads the CPE FPGA, initializes hardware registers, and then scans for the mezzanine type to load the appropriate Framer and PLPE FPGA images. The three DDS (Direct Digital Synthesizer) are then initialized. Next the Interrupt Handlers are installed and enabled. Internal reset of the FPGAs is then performed. Reset of the status data structures is performed.

This system initialization is designed to initialize the system on power up and also when the user switches the protocol using the User Interface. An example is switching from SONET OC-48 to Gigabit Ethernet. Everything stated in the previous paragraph is reloaded and initialized, with the exception of the CPE image, which is common to all protocols. The CPE's function is to pass IP data across the high-speed backplane to the CPE on the adjacent UNAE module.

Status Monitor

The status information for the UNAE card is collected every 100ms by the VxWorks operating system. The source for the 100ms period comes from the CPE, which takes a clock reference and divides it down to generate an accurate interrupt to the I960. The status information is stored in data structures in the VxWorks operating system. The 100ms rate was selected based upon error counter size, rate of incoming data, and ease of calculating error rate information.

The User Interface task is designed to poll the VxWorks system approximately once per second to update the User Interface with the status information.

The status history mechanism is a set of data structures that accumulates the error and alarm conditions over time (time since last reset, as displayed on the User Interface). The "Reset History" button on the UI will clear the data structures and also the counters in the hardware.

PLPE Buffer

The SONET POS was designed with a buffer between the packet stream and the microprocessor. The buffer allows packets to be filtered from the incoming stream and also allows the microprocessor to inject packets into the outgoing stream. This is required for POS for PPP negotiation. The SONET POS PLPE FPGA image has filters to check the receive stream for LCP (Link Control Protocol) and IPCP (Internet Protocol Control Packets) contained in the HDLC framing. These packets are filtered from the stream and redirected to a FIFO implemented on the FPGA. These packets come across the network from the nearest router and at a very low rate (approximately 2 per second). These packets average about 20 bytes in size. Once a complete packet is placed in the RFIFO, an interrupt is generated to the microprocessor for processing the packet in the PPP software. The transmit path works in reverse. The packet is placed into the PLPE

TFIFO and then a bit is set, informing the PLPE there is a packet to be put in the transmit stream. This outlines the physical mechanism and not the complete PPP State Machine software description.

The data packets, filtered in hardware, are sent to the local I960 microprocessor for processing. These packets are identified in the hardware as setup packets, which can be processed without any critical time constraint. Packets that fit into this category include any type of stream negotiation, where the return packet does not have to get there immediately. This includes PPP/HDLC Negotiation and Gigabit Ethernet Auto Negotiation. The purpose is for the processor to handle this type of packet information so the hardware does not have to handle it. This does not include IP processing, since the data stream would overload the processor in an extremely short time.

The majority of the design and implementation was in the PPP Negotiation software and the Gigabit Ethernet Auto Negotiation software. A packet parser, state machines, packet formatter had to be created for this program. Access to debugging these blocks was also implemented.

Packet Processing

There are three major sections for the processing of the negotiation packets. They are the Packet Parser, Packet State Machine, and the Packet Formatter. The sections are different for each of the defined protocols as the packets will differ in size and content for each protocol. The software is written to include all the sections for all protocols supported and the correct ones are called depending upon the unit setup.

Packet Parser

The first step in processing a setup packet is to parse the header to identify what type of packet it is. Once it is identified and the parameters are extracted, the information can then be passed to the Packet State Machine.

It is the job of the Packet Parser to interface with the receive queue memory system to extract the packet of data from the memory, provide the handshaking with the hardware, and to clear the buffer once the data is extracted.

Packet State Machine

The Packet State Machine receives the packet information from the Packet Parser and then identifies the necessary function. For example, in the case of establishing a connection, a return packet is made by the Packet Formatter to be sent on the Transmit link as a response to a query packet. The Packet State Machine is located in VxWorks and not the host. Each protocol has a separate state machine dedicated for the operation of the protocol selected.

Packet Formatter

A response packet is created by the Packet Formatter to respond to a packet query. The response packet is “formatted” into the appropriate header and data content for the

protocol message to be sent. The packet is placed in the transmit queue memory system and the transmit hardware is notified of a message in the queue to be sent. The transmit hardware sends the packet at the next available opportunity and then clear the transmit queue memory.

Mezzanine FPGA

A similar mechanism to the PLPE Buffer is implemented for Gigabit Ethernet. The Auto Negotiation for Gigabit uses TX Config Register and RX Config Register for communicating the capabilities of each end of the link. The mezzanine FPGA hardware will monitor the RX Config Register for a new value received and when the new value is stable for at least three clock cycles, the FPGA interrupts the microprocessor. The Auto Negotiation software then determines the action to take based upon the Auto Negotiation state machine software and will set a new TX Config Register value to be transmitted. This outlines the physical mechanism and not the complete Gigabit Ethernet State Machine software description.

The Auto Negotiation required a timer of 10 to 20 ms to be designed. This timer is implemented in the Gigabit Ethernet PLPE FPGA as a 15ms timer. The software will start the timer and an interrupt will be generated 15ms later. The timer is controlled and used by the Gigabit Ethernet State Machine software.

PPP State Machine Software

The transmission of IP over SONET uses Point To Point (PPP) protocol for transport of packets. This link provides a full duplex bi-directional operation. The PPP protocol uses an HDLC-like framing for encapsulation, a Link Control Protocol (LCP), and Network Control Protocol (NCP).

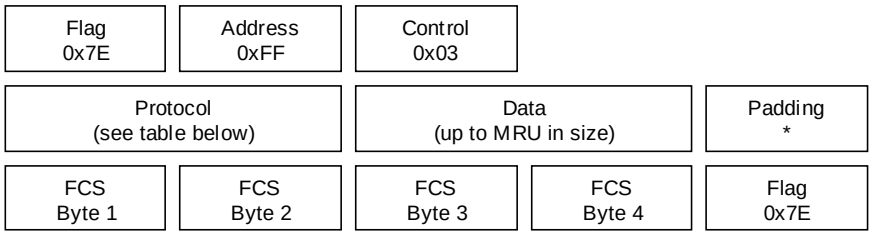
To establish communication over the link, each end must first send LCP packets to configure and test the link. Once the link is established, each end may be requested to authenticate itself.

Once the connection has been established, the NCP packets are sent to configure one or more of the network layer protocols. When all of the network layer protocols have been configured, then datagrams from each network layer protocol can be sent over the link.

The link will remain configured until a LCP or NCP packet is sent to close the link, an inactivity timer expires, or one of the systems shuts it down.

The SONET POS PPP Negotiation software was designed and implemented according to RFC1332, RFC1619, RFC1661, RFC1662, and RFC2023.

The HDLC encapsulation fields are shown below.



The PPP Protocol Field Values are listed below.

Value	Description
0x0001	Padding Protocol
0x0021	Internet Protocol
0x002d	Van Jacobson Compressed TCP/IP (may not be relevant)
0x002f	Van Jacobson Uncompressed TCP/IP (may not be relevant)
0x004f	IP6 Header Compression
0x8021	Internet Protocol Control Protocol (Network Control Protocol)
0x804f	IP6 Header Compression Protocol (Network Control Protocol)
0xc021	Link Control Protocol

Note: Only Protocol field values relevant to IP over SONET are shown in this table. Refer to RFC 1700 for other values.

Link Control Protocol

PPP uses Link Control Protocol or LCP, illustrated in Figure 6 and Figure 7, for allowing two links to establish operating parameters. These parameters include encapsulation format options, size of packets, detecting loop back link, misconfiguration errors, establishing or terminating a link, authentication of the link, and determining when a link is operating correctly or failing.

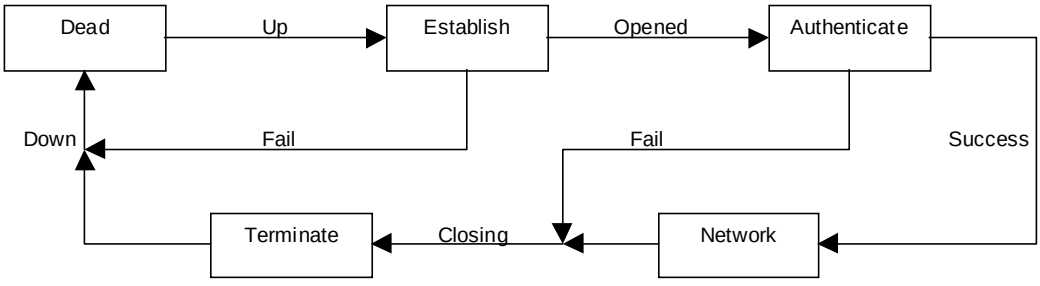


Figure 6: Link Control Protocol Flowchart

	0 initial	1 starting	2 closed	3 stopped	4 closing	5 stopping	6 Req-sent	7 Ack-rcvd	8 Ack-sent	9 opened
Up	2	irc, scr, 6	-	-	-	-	-	-	-	-
Down	-	-	0	tls, 1	0	1	1	1	1	tld, 1
Open	tls, 1	1	irc, scr, 6	3r	5r	5r	6	7	8	9r
Close	0	tlf, 0	2	2	4	4	irc, str, 4	irc, str, 4	irc, str, 4	tld, irc, str, 4
TO+	-	-	-	-	str, 4	str, 5	scr, 6	scr, 6	scr, 8	-
TO-	-	-	-	-	tlf, 2	tlf, 3	tlf, 3p	tlf, 3p	tlf, 3p	-
RCR+	-	-	sta, 2	irc, scr sca, 8	4	5	sca, 8	sca, tlu, 9	sca, 8	tld, scr, sca, 8
RCR-	-	-	sta, 2	irc, scr, scn, 6	4	5	scn, 6	scn, 7	scn, 6	tld, scr, scn, 6
RCA	-	-	sta, 2	sta, 3	4	5	irc, 7	scr, 6x	irc, tlu, 9	tld, scr, 6x
RCN	-	-	sta, 2	sta, 3	4	5	irc, scr, 6	scr, 6x	irc, scr, 8	tld, scr, 6x
RTR	-	-	sta, 2	sta, 3	sta, 4	sta, 5	sta, 6	sta, 6	sta, 6	tld, zrc, sta, 5
RTA	-	-	2	3	tlf, 2	tlf, 3	6	6	8	tld, scr, 6
RUC	-	-	scj, 2	scj, 3	scj, 4	scj, 5	scj, 6	scj, 7	scj, 8	scj, 9
RXJ+	-	-	2	3	4	5	6	6	8	9
RXJ-	-	-	tlf, 2	tlf, 3	tlf, 2	tlf, 3	tlf, 3	tlf, 3	tlf, 3	tld, irc, str, 5
RXR	-	-	2	3	4	5	6	7	8	ser, 9

Figure 7: Link Control Protocol State MachineTable

Definitions:

EVENTS	ACTIONS
UP = Lower layer is up	tlu = This layer up.
Down = Lower layer is down	tld = This layer down.
Open = Administrative Open	tls = This layer started.
Close = Administrative Close	tlf = This layer finished.
TO+ = Timeout with count > 0	irc = initialize restart count

TO-	= Timeout with count expired	zrc = Zero restart count
RCR+	= Receive configure request (good)	scr = Send configuration request
RCR-	= Receive configure request (bad)	
RCA	= Receive configure ack	sca = Send configure ack
RCN	= Receive configure nak/rej	scn = send configure nak/rej
RTR	= Receive terminate request	str = send terminate request
RTA	= Receive terminate ack	sta = send terminate ack
RUC	= receive unknown code	scj = send code reject
RXJ+	= receive protocol reject (permitted)	
RXJ-	= receive protocol reject (catastrophic)	
RXR	= receive echo request or reply	ser = send echo reply

Network Control Protocol

Network Control Protocol (NCP) packets configure the individual network protocols over PPP. For Packet Over SONET (POS), PPP uses Internet Protocol Control Protocol (IPCP as stated in RFC 1332. There are two configuration options that are valid: IP Address and IP Compression Control. The NCP negotiation is performed after the LCP negotiation has completed. Once the NCP negotiation is completed, then the data stream is enabled in the Transmit path.

IP Address provides a way to negotiate the IP address on each end of the link. It is used for either a configuration request or a request that the peer provides the information.

IP Compression Protocol provides a way to negotiate the use of specific compression protocol. The only compression method supported is the Van Jacobson Compressed TCP/IP protocol. Default is no compression and was not implemented for the UNAS.

PPP Negotiation Results

The PPP Negotiation software was successfully implemented and debugged. The UNAE software was tested by using an IXIA OC-48 tester, an Adtech OC-48 tester, a Juniper OC-48 router, and a Cisco OC-48 router. In the case of the Juniper router, it was discovered the PPP Negotiation software would hang in the IPCP state. It was determined that sending an IP address request to the Juniper router would not even generate a reply from the router. This issue did not exist in the Cisco router or the OC-48 testers. The NGI team put a fix in the state machine to circumvent this problem due to the IP address from the IPCP message exchange is not used by the UNAS. This change did not affect the operation of the PPP Negotiation on any other connection.

Operating State

The Operating State design consists of messages passed from the host to the UNAE card

to setup the system properly. The message is a result of the user interaction to establish the correct operating state. These messages will in turn cause a sequence of operations to be performed on the UNAE card, such as loading new PLD images and/or sequence of new hardware register values to establish a new operating state. The User Interface and the hardware must match for correct operation and message passing. An example is if POS is added to the receive path. The POS PLD image will be loaded, new POS status information will be gathered in VxWorks, new POS messages will be sent between the User Interface and the UNAE card, and the POS User Interface will be displayed.

System operation also includes protocol negotiation. This will be an interaction with protocol messages for establishing the connection between the links. The data stream for POS will not be sent until the PPP Negotiation has been completed. Similarly, the data stream for Gigabit Ethernet will not be sent until Auto Negotiation has been completed.

Software Updates

The UNAS system is designed to allow easy software updates for user, field, or developmental updates. This includes User Interface, NT Device Drivers, VxWorks Application Module, and Unasservice.

Debug

Each of the operating systems provides a method of accessing debug information allowing the development team to verify the system design. Tools are written for the NGI program to allow access for hardware development as well as software development.

Shutdown

The UNAS system shut down by closing all shell windows and the UI. The Device Driver can then be stopped at that point. The UNAS is shut down using the Windows Start Menu similar to a PC running Windows.

4.1.2.3 User Interface

The User Interface (UI) is designed under the NGI program to allow the user to setup the system for correct operation and to display status information relevant to the selected protocol. The operation of the User Interface is dynamic as the UNAS is configured. The User Interface, with more than one UI (local and remote), interacts with a common host device driver to send and receive messages to the individual UNAE cards. Selection of a different protocol will require a new screen to appear and the correct status information to be displayed.

The Windows NT User Interface resides on the host system to allow local unit setup. The User Interface status information is updated approximately once per second. Visual C++ Version 6.0 is used to construct the User Interface components.

The User Interface consists of several UI panels or “pages”. Each protocol and data

encapsulation has a separate Tx/Rx page, which includes all setup and status information.

Host User Interface Components

The design of the UNAS User Interface for the NGI program set created some interesting challenges. The first is how to display multiple modules on the User Interface clearly. The second challenge was the dynamic nature of each UNAE card to adapt to multiple protocols. The implementation results are in the following sections.

UNAE Module Selector

The UNAE UI has a series of UNAE card icons along the left hand side for each UNAE module. They allow the user to display the configuration and status for a selected module. Each of the UNAE modules will have an icon and setup pages associated with it. The icon displays the slot number the module is located in (for ease of physical connections) and the protocol the module is set to. An example is one UNAE card configured as SMPTE 292 and the second UNAE card configured as SONET OC-48. The user will see two identical boards when attempting to physically connect the signals. The indicator will let the user know which board is configured as the SONET board and the slot number the module is in. Clicking on an icon will display the setup and status for that module.

Setup Page

The first panel of the User Interface is the "Setup" page. This panel is a tabbed page labeled "Setup". The Setup page allows the user to select the protocol for a UNAE card. The selection of a protocol on the configuration page causes the remaining UI pages to be "swapped" with the new selected protocol specific UI pages.

The Setup tabbed page is always present in the UNAS system regardless of the protocols used.

Protocol Specific UI Pages

Each protocol has a specific page with all Tx setup parameters, Rx setup parameters, and status information such as LOS, bit errors, LOF, etc. The following pages are written to support the UNAS: SONET (OC-3, OC-12, OC-48), Gigabit Ethernet, SMPTE292, SMPTE259, and PPP/HDLC.

These protocol specific tabbed pages are dynamic in nature. If the protocol is changed on the setup page, these protocol specific tabbed pages will be replaced with new one appropriate to the new selected protocol.

Host UI Connection to Device Driver

The host UI connects to the Device Driver with the Microsoft standard "DeviceIOControl" message.

Remote User Interface

The User Interface also consists of a remote user interface to access the UNAS. We implemented three remote UIs using HTML, Win 32, and JAVA. Each of these remote UI designs had strengths and drawbacks.

The HTML design could be used from any type of computer using Windows, LINUX, or UNIX. However, HTML is always a request/send type of interface, where the UI is not updated unless requested by the user. A "push" technology could be developed in the future for HTML to be useable for this type of application.

The JAVA design could also be used from any type of computer running Windows, LINUX, or UNIX. However, JAVA had two issues. The first issue was initialization time of JAVA when used on a SUN workstation. It took several minutes to load the JAVA libraries across the network to run the application. The second issue was the tools used for the JAVA implementation only allowed a UI that had a "primitive" appearance rather than a more polished approach.

The Win32 approach was not as portable to run on any type of computer, only on computers running Windows. However, the strength of recompiling the existing User Interface for remote operation, allowing the unit to be run exactly the same as the host, was important. The conclusion is that most installations would have access to a PC running Windows, so the portability issue became small. This is the approach taken by the NGI Team.

Remote UI Connection to the Device Driver

The host UI connects to the Device Driver with an internally developed "NGIServer.DoCommand" message. This message will communicate with an application running on the host, which listens to a socket interface for messages from the remote UI. The application running on the host is "Unasservice.exe". The messages received from the remote UI will be in the form of "NGIServer.DoCommand" calls and the application will convert them into "DeviceIOControl" calls and send them to the Device Driver.

The remote UI will prompt the user for an IP address and port number to connect to. Once it is connected, it will appear exactly the same as the host UI.

Client and Server Programs

This program includes **UnasService.exe** and **UnasRemoteClient.exe** files. UnasService.exe is a DCOM server providing the service to UnasRemoteClient.exe. UnasService.exe can be running on the same or different machine from UnasRemoteClient.exe. The functionality of UnasService.exe is to deliver the request from UnasRemoteClient.exe to NT device driver and the output of device driver to the remote client.

The primary purpose of UnasRemoteClient.exe is to graphically display the real-time activity of UNAE devices received from UnasService.exe. The connection between UnasService.exe and UnasRemoteClient.exe is via DCOM protocol. The UnasService

proxy/stub object makes the DCOM protocol work. The proxy ensures that parameters are correctly marshaled across the process of apartment boundary and initiates the RPC calls; while the stub ensures that stack is correctly constructed with the un-marshaled parameters sent by the proxy, and that the method is called. COM/DCOM is built on the top of RPC. Any fancy features are realized in the RPC functions.

NGI Web program

The NGI Web program includes NGIWeb.dll (ActiveX controls) and NGIWebExt.dll (ISAPI extension) files. NGIWeb.dll works in the same workspace of Internet Explorer. NGIWebExt.dll works in the same workspace of NT web server, such as IIS 4.0 in this program. When Internet Explorer opens a page with ActiveX control object embedded, it will download and register the object to the local machine, so how to maintain the continuous communication between the ActiveX control and the NGI devices is critical to this program. I finally come out an idea with ISAPI extension. For each request from web browser (actually from ActiveX control), the request will be routed to ISAPI extension (NGIWebExt.dll) residing on IIS web server. The ISAPI extension has every method that the ActiveX control may need. It handles all requests from ActiveX controls based on the format and content of query string it received.

NGIWebExt.dll is a web server extension written in VC++ with MFC library. Basically, when NGIWebExt.dll receives a request, it will deliver the request to the device driver by calling *DeviceIoControl()* system method and get corresponding output from the device driver. The ISAPI extension writes the output into HTML standard stream in a certain format. At the meantime, the program also writes the error message into Application Log file if error occurs.

NGIWeb.dll is written in Visual C++ 6.0 with Active Template Library (ATL) and Microsoft Foundation Class (MFC). The ActiveX control is the COM object with GUI features. The NGI Web program has 10 ActiveX controls (Figure 8 Remote User Interface using DCOMS). All controls are ATL *Lite Composite Controls* with MFC enabled. NGI_MAIN_DLG control is the main control and hosts the other 9 controls that are dynamically loaded and displayed by NGI_MAIN_DLG at run time. A timer is used to continuously update NGI Web display. The timer is set in NGI_MAIN_DLG control and passed into other 9 controls via the interface method.

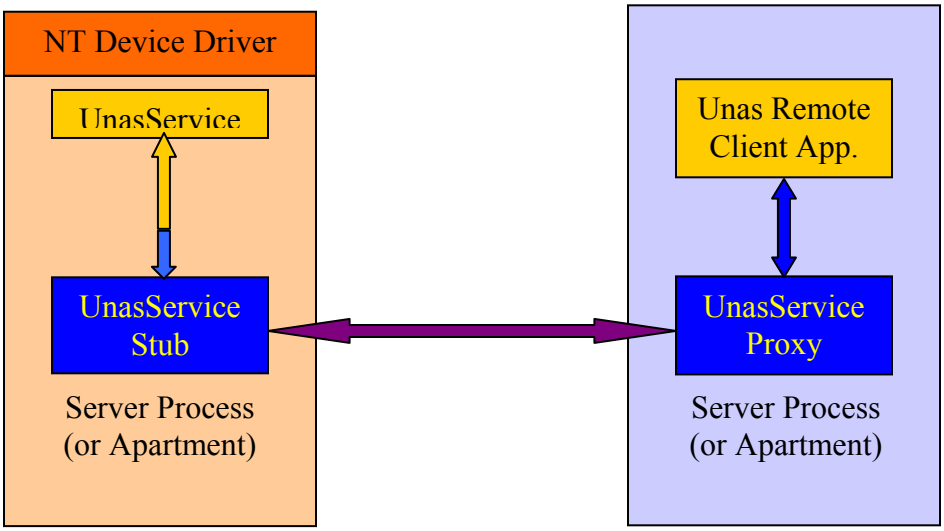


Figure 8 Remote User Interface using DCOMS

Limitations of Client Server Model

The programs are developed with Visual Studio 6.0 on Windows NT 4.0 platform.

1). The programs are developed to support our NGI development team. It does not consider connecting an UNAS outside of Tektronix and another inside Tektronix. The client/server program will not be able to pass the firewall and web proxy unless the proper configurations have been performed on the Tektronix network. The NGI web program is capable of running outside of Tektronix, but authentication through the Tektronix firewall must be verified in advance.

2). Both programs require that user have administrator right or ability to edit system registry on both client and server machines.

3). The program must run on a Windows platform. Windows NT 4.0, 98 or above is preferred. The Internet Explorer 4.0 or above is required for NGI Web program.

4.1.3 System Software build environments

The UNAS system software system consists of several build environments to put together the complete UNAS software system. Each of the software build environments was constructed for the NGI program to build the UNAS software system. Each software build environment will be described in the following sections.

4.1.3.1 VxWorks Core Operating System

The VxWorks core operating system is built using VxWorks 5.2 for the Intel I960 microprocessor. The image is built from the full source code. There were some modifications to the base operating as sent from Wind River. The first modification is to the messaging system, enabling the commands for DMA load of a VxWorks module and to start it. The second modification is the redirection of the shell commands through the

RS232 port on board or through the mailbox registers in the I960 microprocessor to use with the "shell" popup window. A jumper placed in the RS232 port on the UNAE card indicates to the software which of these two paths to use. The installed jumper indicates use of the "mailbox" registers and the jumper not installed will use the RS232 connector. The path used for the "shell" commands is within the main interrupt system of the operating system itself, which is why the core had to be modified rather than including it in the VxWorks Application Module.

There is not any UNAS system information contained in this image. This image is placed into the FLASH ROM by use of the "VxWorks Loader" described below.

4.1.3.2 VxWorks Application Module

The VxWorks Application Module is described earlier in this document. It contains all of the FPGA images for all the protocols, messages for UNAS, hardware specific drivers, control software, PPP negotiation, Gigabit Ethernet auto-negotiation, hardware state information, protocol switching, etc. This module is built by the hardware or software team to test new functionality, either in the FPGA images or the software itself. This module is located on the NT controller hard disk and is loaded by the NT driver into the NT controller local memory. The VxWorks core operating system then performs a DMA operation of the image into the dynamic RAM located on the UNAE board. The VxWorks core operating system then dynamically links the module to the core OS.

4.1.3.3 VxWorks Loader Image

The VxWorks Loader Image, along with the VxWorks Loader Utilities, solves the problem of putting the VxWorks Core OS into FLASH ROM. A new UNAE board will not have any FLASH image and therefore the I960 will not boot up. To solve this problem, the first step is to create an executable image to burn the VxWorks OS into FLASH ROM. This executable image is also a version of VxWorks that will be loaded and run. The VxWorks Loader Image is essentially a VxWorks OS "wrapper" with the VxWorks Core OS (to be placed in FLASH ROM) embedded into it. This "wrapper" image differs from the core in PCI bus address translations, messaging system, etc.

The VxWorks Loader Image is placed into and executed from dynamic RAM on the UNAE card. The size of this "double" image is approximately 4.5 Megabytes. Once executed, it will check the FLASH ROM, erase it if necessary, and then "burn" the new image into the FLASH ROM. The task of placing this Loader Image is performed by the VxWorks Loader Utilities.

There is not any UNAS system information contained in this image. This image is used to burn VxWorks Core OS into the FLASH ROM .

4.1.3.4 VxWorks Loader Utilities Using Cyclone Chassis

The VxWorks Loader Image is built and ready to be placed in a new UNAE board. The VxWorks Loader Utilities run on a "Cyclone" chassis, which is a motherboard manufactured by Cyclone Microsystems. This chassis has several PCI slots and runs

VxWorks. This VxWorks image is different from the other VxWorks images discussed so far. There are utilities included on this host to provide access to the PCI slots for programming the image.

The first step is to use an adapter for Compact-PCI to standard PCI to place the board in the Cyclone chassis. There are switches on the UNAE card that control the microprocessor. There are two modes we use on the UNAE: the first mode is Mode 3, which allows the microprocessor to boot normally from the FLASH ROM, and Mode 0, which places the microprocessor in HALT mode. The UNAE is set to Mode 0 with the switch settings and placed in the Cyclone chassis. The utilities will first program the I960 processor for memory maps, bridge settings, dynamic RAM controller settings, etc. Essentially, this is a "boot strap" process of initializing the I960 system through PCI bus "peek and pokes". Once this is done, the VxWorks Loader Image (the double VxWorks image) is now loaded into dynamic RAM on the UNAE board using the "peek and poke" method. Once the image is loaded into dynamic RAM, a "jump vector" is given to the I960 and the processor is then enabled to run (through a software command). This causes the VxWorks program to execute and "burn" the core OS into FLASH ROM.

There is a major issue that had to be overcome with this approach. The issue is the way the I960 appears on the PCI bus. The PCI registers for the default I960 configuration indicates the I960 is a bridge device and these registers cannot be changed to make it a different device. The setting for the PCI bus address window size is 64K bytes in size. This does not allow the software to pass a 4.5 Megabyte image through this window. On the Compact-PCI mainframe, the operating system is Windows NT 4.0. Windows NT scans the PCI bus at boot time and creates a table for the capabilities of the PCI devices on the bus. Once this table is made, it cannot be changed. This eliminated the use of programming the FLASH ROM on the Compact-PCI platform running Windows NT. Using the Cyclone chassis, the PCI slots are allocated larger blocks of memory, especially if there are no adjacent cards. This allowed the UNAS software team to program the UNAE cards. The drawback and risk is that there is only one Cyclone mainframe working at Tektronix.

There is not any UNAS system information contained in the utilities. The utilities are used to load the VxWorks Loader Image into the UNAE dynamic RAM .

4.1.3.5 VxWorks Loader Utilities Using LINUX

The UNAE cards have their FLASH ROM programmed using the VxWorks Loader Utilities run on a "Cyclone" chassis. The issue with the Cyclone chassis is that there is only one of them working at Tektronix. To eliminate the risk to the program, we developed an alternative to the Cyclone chassis. This is a regular PC running Linux with some modifications. The first is the VxWorks Loader Utilities were ported from VxWorks to the LINUX operating system. The second was reserving approximately 6 Megabytes of memory in the PC (outside what LINUX thought the memory space is) and then setting the PCI bus to use this address space, essentially creating a mapped window into the PCI bus. The program was then able to load the VxWorks Loader Image into the

UNAE dynamic RAM and execute the utilities.

There is not any UNAS system information contained in the utilities. The utilities are used to load the VxWorks Loader Image into the UNAE dynamic RAM

4.1.3.6 Windows NT Device Driver

The Windows NT 4.0 Device Driver is built using the Microsoft Device Driver Kit (DDK) and Microsoft Software Development Kit (SDK) packages. The final built image is "unas.sys".

4.1.3.7 Windows NT User Interface

The Windows User Interface is built using Visual Studio C++ 5.0 or better. The file image for the User Interface is "NGI.exe".

4.1.3.8 Windows NT Remote User Interface

The Windows Remote User Interface is built using Visual Studio C++ 5.0 or better.

4.1.3.9 UNAS Service Task

The UNAS Service Task is started on the host platform to provide the connection for the remote User Interface to operate. It takes the commands from the remote UI and translates them into NT Device Driver calls. This is started from the Windows "Services" dialog box. The image file is called "unasservice.exe" and is compiled using Visual Studio C++ 5.0 or better.

4.2 BRAD ASIC

4.2.1 Design objectives

The BRAD is intended for fiber optics communications applications. On the receive side, the BRAD accepts a serial data input stream, recovers clock and data, and demultiplexes it. On the transmit side, the BRAD accepts data from a parallel interface and multiplexes it into a serial output stream. The BRAD generates clocks for serial and parallel data outputs, and clocks the parallel input.

The project was undertaken to provide these features not generally available in commercial transceiver components for optical communications:

1. **Burst mode** allows the receiver to instantly start acquiring data, without losing a single bit, when a new transmission starts, using known protocol. Most receivers have phase locked loop (PLL) clock recovery circuits, which need a lock-in period before data can be acquired.
2. **Bit rate agile** transmitter and receiver clock circuits. Clocks may be programmed to any serial rate from 150 Mb/s to 3 Gb/s.
3. **Programmable parallel word widths** of 8, 10, 16, 20 and 32 in receiver and transmitter serial-to-parallel and parallel-to-serial converters, in order to accommodate multiple protocols.

4. **A receiver transition counter** helps estimate the bit rate of a received data of unknown protocol.

4.2.2 Summary of Work

4.2.2.1 CMOS feasibility study

It was the objective of a short design study to verify feasibility of realization of the BRAD design using CMOS technology. The ASIC Advanced Development Group at Tektronix has world class design skills, particularly in analog and high speed digital such as global clocks. A senior engineer addressed the problem assuming CMOS processes available to Tektronix at the time and for which well developed models were available. A 0.25u process from IBM was well understood while models for a 0.18u process from National Semiconductor were being integrated into the design group. Models for these two processes were used for simulations.

According to the SONET specification, the longest data pattern without transitions is 384 bits (following the A1A2 bytes, no scrambling). The technical challenge addressed was to maintain low enough jitter to perform clock and data recovery of 384 bits at 3Gbps. Burst capture was not optimized in this study.

Traditionally CMOS devices are used in a lower range of frequencies than Bipolar: Low surface mobility of carriers and high gate capacitance contribute to a relatively slow RC time. Parameter variations between adjacent devices complicate use of CMOS in analog applications. High gate capacitances and full CMOS voltage swing of operation create noise on power, ground and substrate, generating signal jitter. The combined effect determines maximum data rate of a data acquisition system. Several circuit types were considered for the fast receiver front end, including:

- A three-oscillator circuit (one VCO and two gated VCO's – GVCO's) proposed by Banu[iv]
- Addition to this of a phase-correction circuit to reduce jitter uncertainty using a pulse-gated receiver (PGVCO's)
- Interleaved receiver using two PGVCO's and no master VCO
- Receiver with oversampling by 4

Conclusions were that a 3Gbps non-oversampling receiver can be built with margin on 0.18u CMOS process. It was supposed that the most efficient architecture is interleaved PGVCO with a phase error correction circuit. Existing data indicated that 3Gbit/s over-sampling by 4 receiver with phase correction can be built on .25u CMOS process. Over 3GHz bandwidth for a transmitter can be achieved on either process by combining DLL (delay locked loop) and oscillator - based output stages. These conclusions were drawn with limited experience with the accuracy of 0.18u models at the time.

4.2.2.2 Use of Silicon-Germanium Bi-CMOS process

The maximum bit rate of 3 Gb/s is right at the upper end of what the present CMOS processing capability can be expected to handle. In order to operate at this speed, CMOS

circuits usually need a fair amount of tuning and delay compensation. While this might have worked well at any one fixed bit rate, it was felt that it would be risky, given the variable bit rate requirement of this project. In addition, the BRAD was at the time considered to be a gating element in the system. If the CMOS version did not perform at at least 2.5Gbps the remainder of the system would not operate. This is why a SiGe hetero-junction bi-polar process was chosen.

The combination of the Burst mode and Bit Rate Agile features required development of a new approach for the receiver. The voltage controlled oscillator (VCO) and clock recovery circuit (CDR) described below made these features possible.

4.2.2.3 VCO

The VCO is a ring oscillator of two ECL delay elements with programmable delay, and inverting feedback. Each delay element uses a string of buffer delays, and interpolates between taps in an analog fashion to include 0-2 buffer delays, plus the single buffer delay of the pick-off. The oscillation period is thus four element delays, and the outputs of the two buffers provide two versions of the clock, with one-quarter period phase difference, as required by the CDR. The programmable delay allows one octave of clock programmability, about 1.6-3.2 GHz. Two version of this oscillator are used, to insure the full octave of coverage, with temperature and production variations. In this frequency range the oscillator outputs are used directly. D flip-flop binary dividers with master and slave outputs are used to provide the two phase outputs at all lower frequencies. Division by 2, 4, 8 or 16 provides continuous frequency coverage down to below 150 Mb/s, as required. Finally a divided-by-32 clock is phase locked to the external frequency reference. The reference frequency of $1/32^{nd}$ of the oscillator, together with the programmable binary ratio, defines the frequency of the receiver CDR. An similar clock generator provides the transmitter clock, although the transmitter clock does not need the two-phase outputs.

A simplified concept of the circuit is illustrated Figure 9. More detailed schematics for both long delay and short delay versions have been previously provided in quarterly reports.

The differential control voltage input is applied to the input terminals ‘pfreq’ and ‘nfreq’ while the center pin ‘cfreq’ is kept at the common mode average between the two. As the voltage is varied the currents in transistors Qleft, Qcenter and Qright vary as illustrated inFigure 10. The output is thus the sum of proportioned signals with 0-1-2 buffer delays.

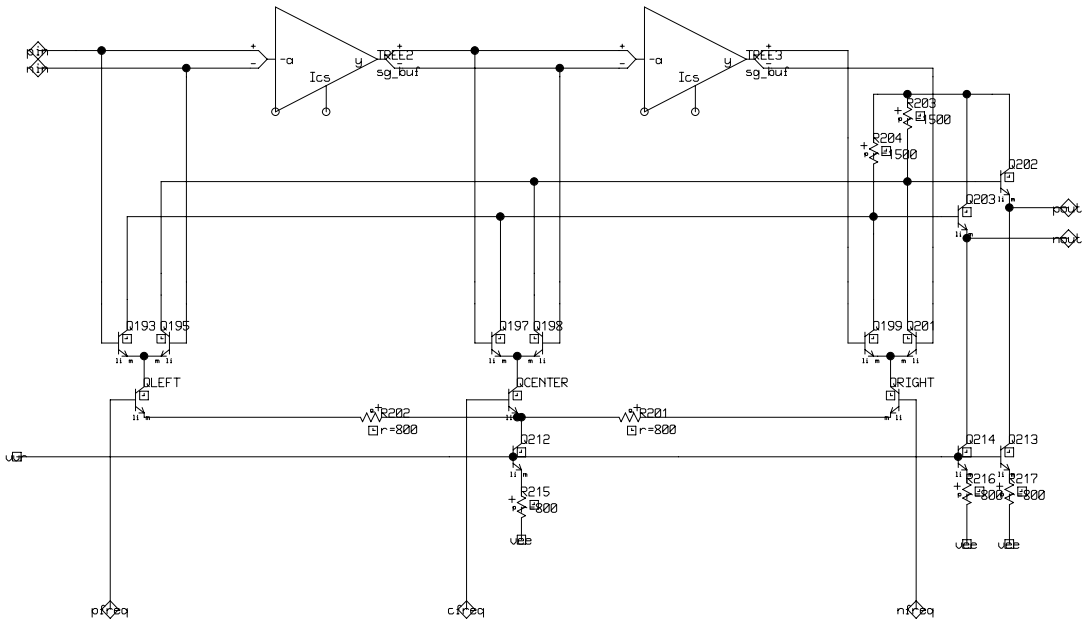


Figure 9 VCO for phase locked loop (PLL)

Figure 11 shows the two-phase output of an oscillator versus control voltage swept with time. The active range is from 3 to 8 nS.

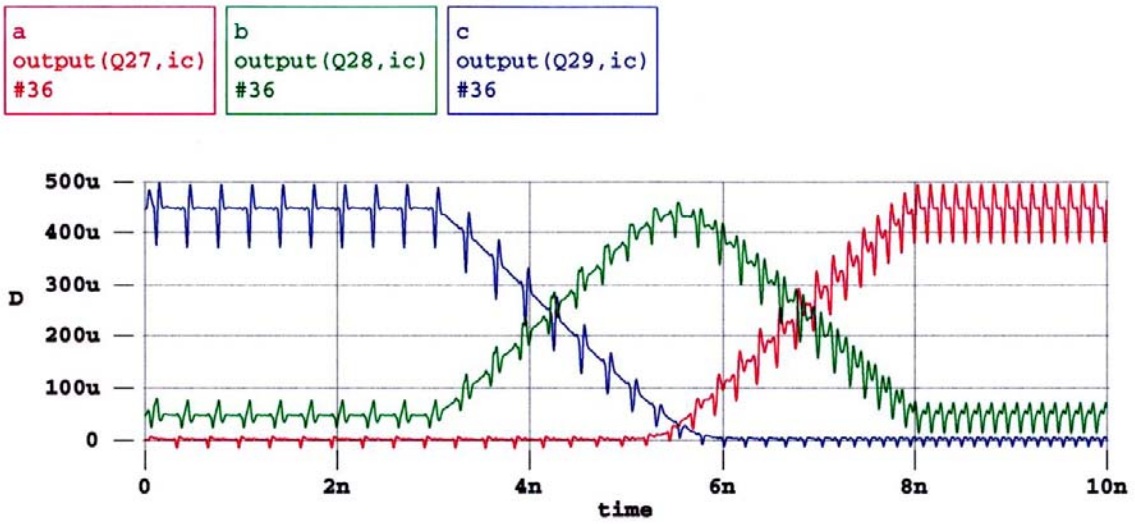


Figure 10 Transistor tail currents in VCO. a = Qright; b = Qcenter; c = Qright

Figure 10 illustrates the tail currents in transistors Qleft, Qcenter, and Qright of Figure 9a, as the control voltage is swept with time as in the Figure above. The active range is

from 3 to 8 nS.

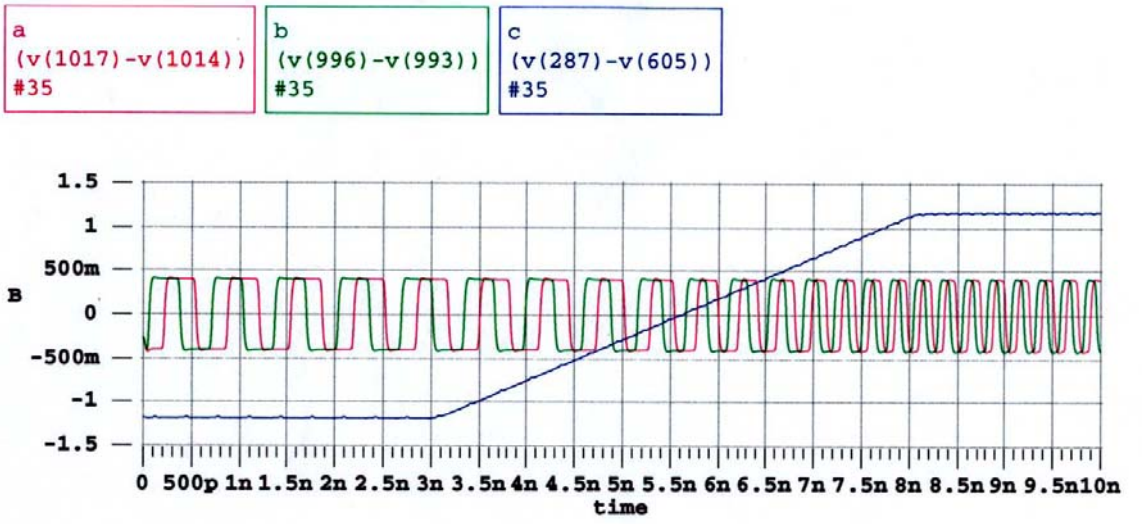


Figure 11 Two phase VCO output. a = phase 2; b = phase 1; c = control voltage

CDR- Key Feature for Burst Mode Bit Rate Agility

Instant acquisition burst mode required a new clock and data recovery approach. A different CMOS approach was investigated first, but at the high 3 Gb/s data rate it was felt that a high performance bipolar process was safer. It was implemented using the IBM SiGe process and cell library developed by the High Frequency Design group at Tektronix. See technical details below.

Simulation results demonstrate instant burst mode clock recovery and glitch free phase correction of the recovered clock as the phase of the input signal slowly drifts with respect to the local VCO clock.

The basic block diagram is shown in Figure 12. It is thought to be a low risk approach, as it is built almost entirely from standard digital building blocks. The only exception is the tunable voltage controlled oscillator, VCO in the phase locked loop. Even the VCO is built as a ring oscillator from emitter coupled logic buffers, but of necessity has a tunable analog feature, which makes it a non-standard cell. In order to guarantee the full one octave tuning range at all temperatures, and for all production variations, two VCO's are used. A standard 2:1 multiplex switch is used, under software control, to select which oscillator is used at any one frequency.

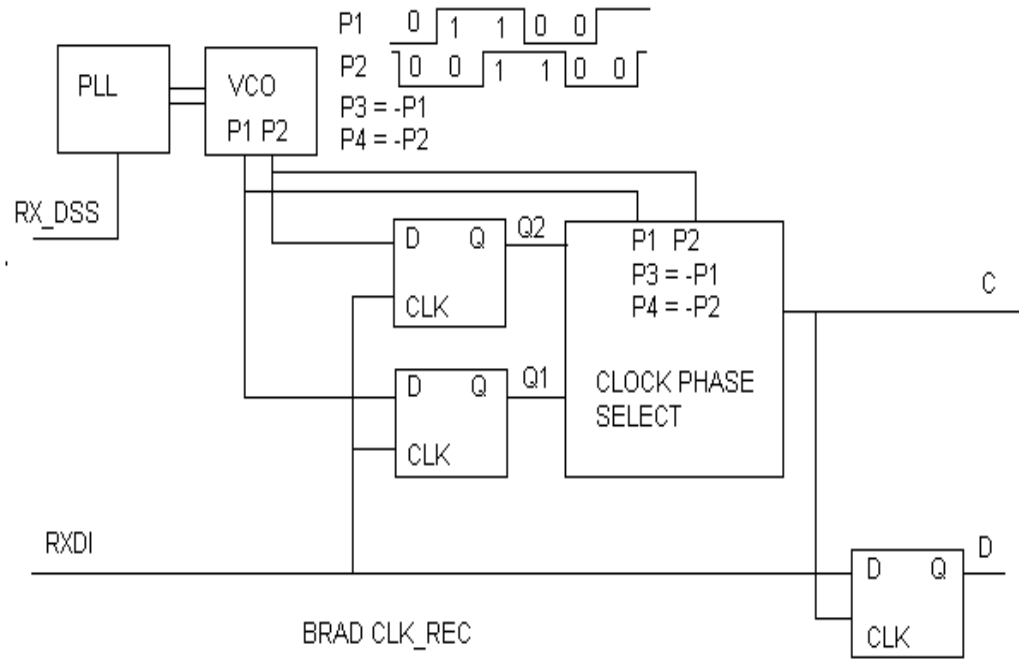


Figure 12 BRAD Clock Recovery Circuit. RXDI = Signal input; RX_DSS = Reference frequency input; C = Recovered clock output; D = Recovered data output

The RX_DSS input controls the frequency of the VCO via the phase locked loop, PLL. The VCO, a two stage ring oscillator, has two differential outputs in quadrature phase, P1 and P2. Two more phases, P3 and P4 are implied as the inverse of these.

The two VCO outputs, P1 and P2, are sampled by the two flip-flops, clocked by transitions of the input signal RXDI. The samples are held as Q1 and Q2. The two flip-flops should ideally be clocked on all positive and negative edges to extract all available timing information from the input, although this is not required. It is possible to use only positive edges, or only negative edges. The present design uses all positive and negative edges.

The samples, Q1 and Q2, define how the input signal phase is related to the phases P1 – P4, as illustrated to the right of VCO in Figure 12. The "1"s and "0"s on the P1 and P2 waveforms are the values of Q1 and Q2 if sampled during that time.

The CLOCK PHASE SELECT circuit uses Q1 and Q2 to select the clock phase, of P1-P4, which fits the timing of the input transition best. That phase now becomes clock C.

If the VCO is perfectly matched in frequency to the input signal clock frequency, the selected phase will not change. When there is a mismatch, the selected phase will cycle 1-4 over and over again, either backwards or forwards, at the beat frequency of the VCO versus the input signal. It is therefore a requirement that the VCO must match the incoming data rate well enough that it drifts less than one-quarter cycle during the longest period of no transitions of the input signal. In SONET OC-48 this could be the 384 UI

immediately following the A1A2 framing signal. These 384 bits are still unscrambled, and the content is uncertain. Thus, the VCO frequency must match incoming data rate better than 1/1546, i.e., about 500 ppm. Reference oscillators sold for SONET applications meet this requirement.

The CLOCK PHASE SELECT circuit is designed to interpolate between adjacent phases if there is meta-stability in either Q1 or Q2, in order to make it meta-stable proof. The circuit tolerates that one or the other is in a meta-stable condition, but never both at the same time.

When there is a clock phase change, the circuit is designed to change during a time when the new phase is equal to the old phase in order to avoid generating glitches.

The jitter of the recovered clock will have a peak-to-peak value of 0.25 UI due to the phase select circuit. While this would have been a problem if the recovered clock were to be retransmitted as a line protocol clock, the recovered clock is only used internally to recover the data. As long as the data is correctly recovered, the jitter of the clock is not a concern.

Note that the PLL and VCO are the only tunable or analog components in this approach. The tunable frequency range is <1.5 GHz to >3 GHz, i.e. greater than 2:1. All lower input clock frequencies will be divided down from this frequency.

The circuit represented by the building blocks in Figure 12 was simulated in ADS, our in-house version of SPICE with schematic capture. All circuits were designed at the transistor level using minimum geometry transistors and “presrp” resistors in the IBM SiGe process. Standard ECL implementations were used for each block. The current trees used 0.5 mA standing current, as did the emitter follower pull-downs. The waveforms in Figure 11 demonstrate that the recovered clock is changing phase as necessary in a glitch free manner. From 0 to 5 nS the local oscillator is running slightly slower than the input signal frequency, and from 5-10 nS it is faster. Note the phase corrections in the recovered clock as it drifts relative to the input signal.

The waveforms in Figure 13 demonstrate feasibility of the concept. In trace a, VCO is too slow 0-5 nS, and too fast 5-10 nS. Hence the adjustments. In b, the recovered clock note short cycles at 1.8, 3.8 and 5.4 nS, and long cycles at 5.8, 7.4 and 9.4 nS. Each immediately follows a change in Q1 or Q2.

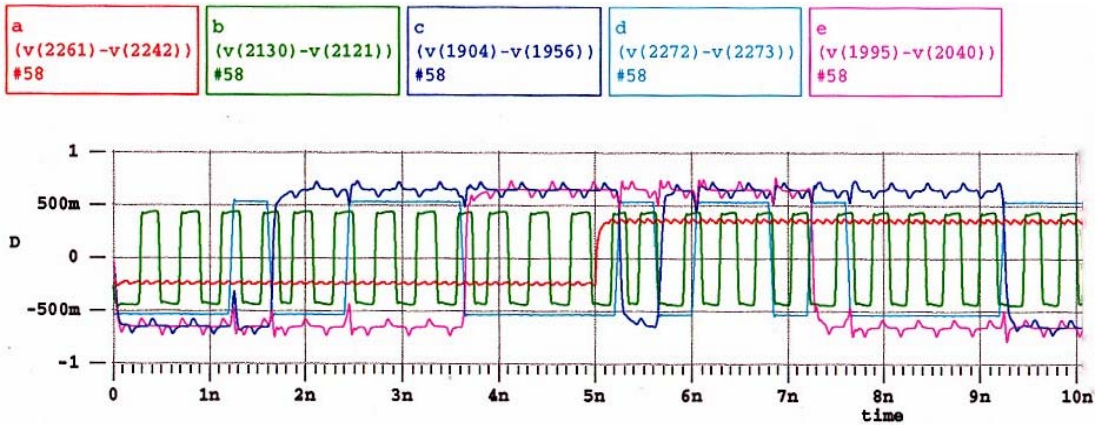


Figure 13 Glitch-less Clock recovery demonstrated. a = VCO control voltage; b = Recovered clock; c = Phase-select bit Q1; d = Input signal; e = Phase-select bit Q2

Serial-to-Parallel and Parallel-to-Serial converters

Parallel word widths are programmable to 8, 10, 16, 20 and 32. It would have been simple to use 1x32 bit serial-parallel shift registers for these, with programmable dividers to control the parallel shifts at the required intervals. However, at the highest speeds it would have been difficult to guarantee clock-data timing over the length of the shift register when doing the parallel shift on the fly. The circuits used have shorter shift registers, and therefore easier to maintain clock-to-data timing. The reader should find the circuits simple and straightforward to understand if the control register section of the specification and the ADS schematics are studied side by side.

4.2.2.4 Limitation of design

The CDR used in the BRAD was selected because it is capable of instantly recovering Clock and Data in known data rate burst mode, without a lock-in period. This capability comes at a price.

The recovered clock will have 0.25 U.I of jitter added to the incoming jitter by the ¼ cycle phase selection feature of the CDR. This is acceptable in the intended application, because the clock is only used to recover data and to generate the lower rate parallel clocks. However, this method of clock and data recovery would not be acceptable if the recovered clock were to be used to re-transmit SONET signals, for example. The addition of 0.25 U.I. of jitter would violate jitter transmit specifications.

If it were necessary to use the recovered clock as a re-transmitted SONET (or other protocol) clock, there would be a couple of options. The simplest one is to use the recovered clock as a reference for a phase locked loop with limited loop bandwidth. The bandwidth has to be low enough to limit jitter to within the specifications for the protocol in question. The line loop-back in the BRAD uses such a feature. The transmit clock in

line loop-back is referenced to the recovered clock via a phase-locked loop. If the instant-lock and re-transmit feature is required, a more complicated method is possible. In Figure 12, the VCO would be producing sine and cosine waves instead of square waves ¼ cycle apart, analog track and hold circuits would replace the flip-flops to determine Q1 and Q2, two four quadrant multipliers would replace the 4:1 multiplexer, and finally a difference amplifier would subtract the outputs of the two multiplier outputs.

4.2.3 Results

The following are characteristics of the BRAD design.

The BRAD is intended for fiber optics communications applications.

On the receive side, the BRAD accepts a serial data input stream, recovers clock and data, and demultiplexes it. On the transmit side, the BRAD accepts data from a parallel interface and multiplexes it into a serial output stream.

The BRAD generates clocks for serial and parallel data outputs, and clocks the parallel input.

The receiver and transmitter operate independent from each other at any frequency from 150 MHz to 3 GHz. Multiplex and de-multiplex ratios are independently programmable to 8, 10, 16, 20 or 32. There is a transition counter to estimate data rate of received signals of unknown protocol for setting the initial frequency of the clock recovery circuit.

The Clock and Data Recovery in the receiver will recover a burst data of a known rate without losing any initial data while locking in.

Loop-back is built into the BRAD in both directions. Loop-back in each direction may be used or not used independent of the other.

The control registers have separate resets for receiver and transmitter circuits. This allows one to be reset while the other is passing payload, without disturbing it.

The BRAD design was completed through preliminary layout, including placement of circuit blocks and simulation of all analogue and high speed subcircuits. Performance at 3+Gbps was clearly realizable. The BRAD development was frozen at that point, due to growth of both NRE costs and schedule for completion.

4.2.4 Conclusions

The goal of the BRAD design was to achieve several features not generally available in commercial communications chips:

1. **Burst mode** allows the receiver to instantly start acquiring data, without losing a single bit, when a new transmission starts, using known protocol. Most receivers have phase locked loop (PLL) clock recovery circuits, which need a lock-in period before data can be acquired.
2. **Bit rate agile transmitter and receiver** clock circuits. Clocks may be programmed to any serial rate from 150 Mb/s to 3 Gb/s.

3. **Programmable parallel word widths** of 8, 10, 16, 20 and 32 in receiver and transmitter serial-to-parallel and parallel-to-serial converters, in order to accommodate multiple protocols.
4. A receiver transition counter **helps estimate the bit rate** of received data of an unknown protocol.

The present design includes all four features. It is significant that, had this ASIC been realized on the intended time schedule, it would have been to the authors' knowledge, the first 2.5Gbps burst receiver that recovered both clock and data with no loss of data. In addition, simulations have shown that the front-end design is scalable to 10Gbps (with additional design effort for performance).

In addition, although the CMOS approach did pose significant risk, the cost of completing the CMOS design would have been more in line with program resources. The decision to use SiGE for the BRAD was based substantially on a plan to use a very economical multi-project SiGe fabrication option with the foundry. When this option was withdrawn, the fabrication cost became a major factor.

In retrospect, the risk of the CMOS approach failing due to performance would have been manageable, since a substitute multirate (albeit not continuously variable rate) commercial CDR ASIC was found that allows SONET OC-3, OC-12, OC-48 and Gigabit Ethernet transceiver functions.

The decision not to complete the design/fabrication/test/package/characterize process for the BRAD also impacted other elements of implementing the multiprotocol UNAE as planned. The proposed architecture would support two Gigabit Ethernet clock rates (1000Mbps and 1250Mbps due to 8B/10B coding) and also multirate SONET clocks (multiples of 155.25Mbps) in the same optical mezzanine board. The SONET clocks are not multiples of GbE clocks. This fact led to a redesign of the optical mezzanine board to include two local oscillators and substantial clock selection and distribution code in a large FPGA. Details are presented in the Summary of Work section for Gigabit Ethernet.

The BRAD was also the critical element in the plan for the UNAE to discover protocols, since the BRAD could communicate the exact input bit rate to the rest of the system. Without the BRAD the protocol discovery is disable Mezzanine boards

4.2.5 Design objectives

The optical mezzanine provides a physical layer interface for the UNAE system when it is transmitting packets using SONET or Gigabit Ethernet (GbE) protocols. The optical mezzanine attaches to the UNAE main board via the standard mezzanine connectors. The circuitry contained on the optical mezzanine provides the functions of electrical to optical signal conversion, clock and data recovery, serialization / deserialization, signal level conversion, data bus bit width conversion, and Gigabit Ethernet specific physical medium attachment functions. Additionally, for interfacing to an optical router, an electrical interface is provided.

The video mezzanine uses commercial chips to provide serialization and deserialization

and framing functions for either high definition (HD) or standard (SD) digital video. The receive side provides frame delineation for SDI data and the transmit side multiplexes a ten or 20 bit data stream and adds SDI frame boundaries.

4.2.6 Optical mezzanine summary of work

4.2.6.1 Board architecture

Two functional versions of the optical mezzanine board were built. The first, g2469xb, had much the same design as the later card, but had no clock oscillator built onto the board (reference clocks for the optical input/output were generated by the clock synthesis chips DDS1 and DDS2 on the UNAE board and passed up to the mezzanine. It was later discovered that these clocks did not have low enough jitter for SONET interfaces, and a SONET clock oscillator was retrofitted to the board. The second version of the board, g2469xc, included two selectable clock oscillators for SONET (155.52 MHz) and GbE (156.25 MHz). The second version of the board also contained a much larger FPGA with sufficient resources to implement the GbE Physical Medium Attachment / Physical Coding Sublayer (PMA/PCS) logic and additional external connections for an all-optical router packet label. An addressable processor bus was also extended up to the mezzanine.

A block diagram of the components on the optical mezzanine are shown in Figure 14 below. Network inputs and outputs on the left side of the diagram are switchable between electrical and optical ports. Optical ports are 1.3 μm single modes SC connectors and electrical ports are single-ended PECL AC-coupled to SMA connectors. Data rates through this interface are 155.52/622.08/1250/2488.32 Mb/s, selected via RATESEL control signals to the AMCC SERDES chip.

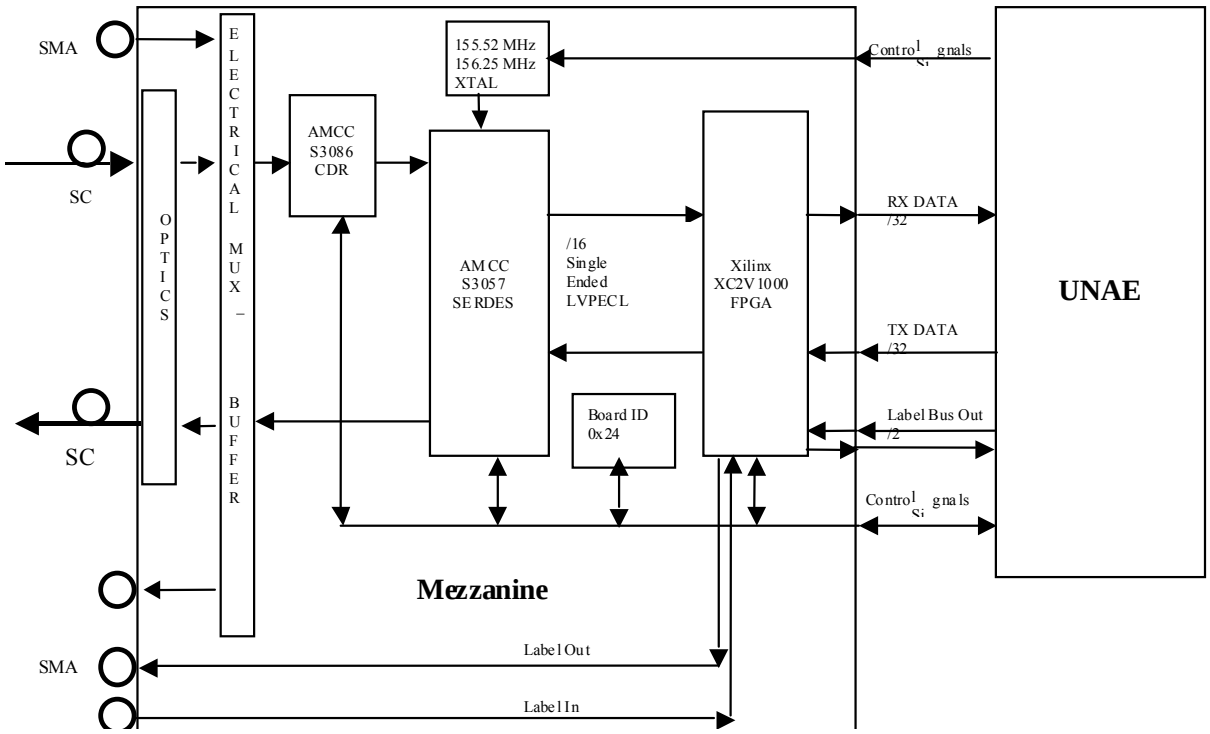


Figure 14 Diagram of optical mezzanine components and connections

4.2.6.2 PMA/POS FPGA design

The PMA FPGA located on the optical mezzanine provides data path conversion functions when operating in a SONET mode. In the receive direction, the FPGA converts the 16-bit wide LVPECL data bus into a 32-bit wide LVTTL bus for transmission across the mezzanine connector to the Framer FPGA on the UNAE main board. A divide by two version of the receive clock is also provided to the UNAE main board. Likewise in the transmit direction, the PMA FPGA accepts the 32-bit LVTTL data bus from the Framer FPGA, and converts this to a 16-bit LVPECL bus to be driven to the S3057. The transmit clock is taken from the PCLKP/PCLKN output of the S3057, divided by two, and driven down to the UNAE main board, which then distributes this as the transmit data path clock.

The data bus width and signal level conversion functions described above are provided by the PMA FPGA whether in a SONET or in a Gigabit Ethernet mode (clocking is handled slightly differently). In order to support Gigabit Ethernet operation, however, additional data path and clock functions are required. The mode selection determines whether the additional data path functions are bypassed (in a SONET mode), and determines which of the transmit and receive clocks (a simple divide by 2 for SONET, or

an internally synthesized derivative of the original clocks for Gigabit Ethernet) is passed on to the UNAE.

The Gigabit Ethernet mode of operation requires the use of 8B/10B encoding for the Physical Coding Sub-layer. In order to support this mode of operation, the PMA FPGA must, in addition to the 8B/10B encoder and decoder, perform the necessary path width and clock rate conversion for exchange of 4 (8-bit) words at a time on the mezzanine interface, and exchanging 10-bit code words 16 bits at a time at the S3057 data interface. Because of the 25% overhead involved with the Gigabit Ethernet PCS function, the line rate is 1250Mbps. This means that the 16-bit interface to the S3057 will operate at 78.125MHz. Both sides of the 8B/10B encoding function operate at 125MHz (resulting in 1000Mbps/1250Mbps). This clock frequency is an 8/5 multiple of the 78.125MHz clocks provided by the S3057, and must be synthesized internally by the FPGA. This is accomplished by utilizing the DCM frequency synthesis features found in the Xilinx Virtex2 parts. Additionally, because of the non-integer multiple nature of the 8/5 clock multiplication, the two sections of the data paths must be treated as separate clock domains. Crossing the boundary between these domains, as well as providing an 8B/10B path width conversion, is a pair (transmit and receive) of FIFO's.

For a more detailed description of the functions of the PMA FPGA, refer to the "PMA FPGA Design Specification" in the Appendix. In support of the CDR, the RATESEL lines controlling the line rate for the Transceiver are also brought into the PMA FPGA. These are the decoded to provide the proper BAND_SEL value used to control the CDR.

4.2.6.3 UNAE interface

Three 80-pin mezzanine connectors provide the interface to the UNAE main board. The signals comprising this interface fall into four categories: transmit data, receive data, control signals, and clock signals. The transmit and receive data busses are both 32-bits wide, and carry data between the PMA Framer FPGA's. The control signals are a group of signals between the mezzanine hardware and the Framer FPGA's, which provide various control, configuration, and monitoring functions. Each of these signals' direction and (when not drive by mezzanine HW) state is controlled by the mezzanine Control Registers within the Framer. In general, the CTL<16:0> signals interface directly with the Transceiver and CDR IC's, and the CTL<29:17> interface directly with the PMA. The clock signals are: CCLK, CLK78, TCLK, and RCLK. CCLK provides the configuration clock for the PMA FPGA. The TCLK and RCLK are the transmit and receive clocks respectively, and are both driven from the PMA FPGA to the UNAE main board, where they are distributed as the data path clocks. These clocks will run at 77.76MHz for SONET OC-48 operation, and at 31.25MHz for Gigabit Ethernet operation. The CLK78 signal is used as the REFCLK input for the S3057. In the event that a single crystal oscillator can be used for multiple data rate functions with the new CDR chip, this clock

4.2.7 Video mezzanine summary of work

4.2.7.1 HD video mezzanines

Two separate mezzanine cards for SMPTE 292M uncompressed high definition video transport were previously built by Tektronix as part of an HD video over ATM project. These mezzanine cards, an HDIN video deserializer and HDOUT video serializer, were used with minor changes for this project. Modifications were necessary to make the outputs from the 5 V boards compatible with the 3.3 V FPGAs on the UNAE. The cards each use one of the 80-pin mezzanine connectors discussed above for the optical mezzanine.

1. Changes to the HDIN mezzanine (u0393xa) are as follows:
2. Add 100 ohms series resistors to board ID outputs to limit input current to the UNAE.
3. Remove power to the “silicon serial number” chip.
4. Add 100 ohm series resistor to lock indication output to limit input current to the UNAE.
5. Change pullup on board ID chip select from 5 V to 3.7 V.
6. Disconnect SEL1 input.
7. Put limiting diode to 3.3 V on FFIN input.
8. Disconnect PAR output.

9. Changes to the HDOUT mezzanine (u0392xa) are as follows:
10. Add 100 ohms series resistors to board ID outputs to limit input current to the UNAE.
11. Remove power to the “silicon serial number” chip.
12. Connect the clock input from the UNAE board to the mezzanine clock output line through a 39 ohm series resistor.
13. Change pullup on board ID chip select from 5 V to 3.7 V.

4.2.7.2 SD video mezzanines

Two separate mezzanine cards for SMPTE 259M uncompressed standard definition video transport were previously built by Tektronix as part of an SD video over ATM project. These mezzanine cards, an SDIN video deserializer and SDOUT video serializer, were used with minor changes for this project. Modifications were necessary to make the outputs from the 5 V boards compatible with the 3.3 V FPGAs on the UNAE. The cards each use one of the 80-pin mezzanine connectors discussed above for the optical mezzanine.

Changes to the SDIN mezzanine (u9d2379-00) are as follows:

1. Increase resistance of serial termination resistors on the parallel video outputs to 300 ohms to limit input current to the UNAE.
2. Remove power to the “silicon serial number” chip.
3. Change the power supply and pull-up resistors to the board ID chip to 3.3 V.

4. Change the board ID resistors so that the ID register reads 0x19.
5. Changes to the SDOUT mezzanine (u9b2527-00) are as follows:
6. Connect the clock input from the UNAE board to the mezzanine clock output line.
7. Remove power to the “silicon serial number” chip.
8. Change the power supply and pull-up resistors to the board ID chip to 3.3 V.
9. Change the board ID resistors so that the ID register reads 0x39.

4.3 UNAE board

4.3.1 Design Objectives

The Universal Network Access Engine (UNAE) was conceived as a card level, programmable module capable of receiving serial data, extracting desired information and passing key elements of the stream to a parallel interface. It should have the capability to process the information sufficiently to perform format or protocol translation. Similarly, the UNAE is expected to transfer and translate information from the parallel interface to one or more serial interfaces. The universal aspect is derived from its being rapidly programmable at several logical steps in the data flow and its accommodating any of several modular media attachment interfaces. The UNAE module throughput must be sufficient to handle data in the 100Mbps – 3Gbps range in real time.

4.3.2 Summary of Work

4.3.2.1 Introduction

The architectural motivation for the UNAE board design is included in section 3.2.2; and the functional layout is displayed in Figure 4. The UNAE is a protocol and data format processing engine card that is intended to rely on modular mezzanine cards to act as physical media attachment interfaces. For example fiber, coax and custom data and power interface cables such as IEEE-1394 Firewire could be accommodated without physical changes to the UNAE base card. In a system, two or more UNAE cards can communicate across a UTOPIA-3 or SPI-3 parallel interface realized through use of the P1/P3/P5 back plane connectors of the cPCI platform. In this way two UNAE cards with attached mezzanine cards can act to convert data between two dissimilar protocols. Although this architecture supports UTOPIA3 for point to point Link layer transport of OC-48 ATM cells, reference will be made henceforth to an SPI-3 interface[v] which supports packet transport at OC-48. Alternatively, user data in a serial format could be assembled into an appropriate protocol for transport and routing, as described below where two video formats have been mapped into various protocols. Figure 15 shows a UNAE board with optical mezzanine attached. The board is a 6u size cPCI card. Major elements such as the backside P1, P3, P5 connectors, the three major FPGA’s and FIFO’s may be seen. As seen in Figure 4 and Figure 15, the functioning engine is comprised of a mezzanine board, Framer, physical layer protocol engine (PLPE) a FIFO and cell/packet engine (CPE). The UNAE is a bi-directional interface: The transmit and receive sides are independent, except when one of the loopback modes is enabled. The UNAE provides a full duplex interface, providing the received and transmitted data on the duplex interface

are using the same protocol or data format.

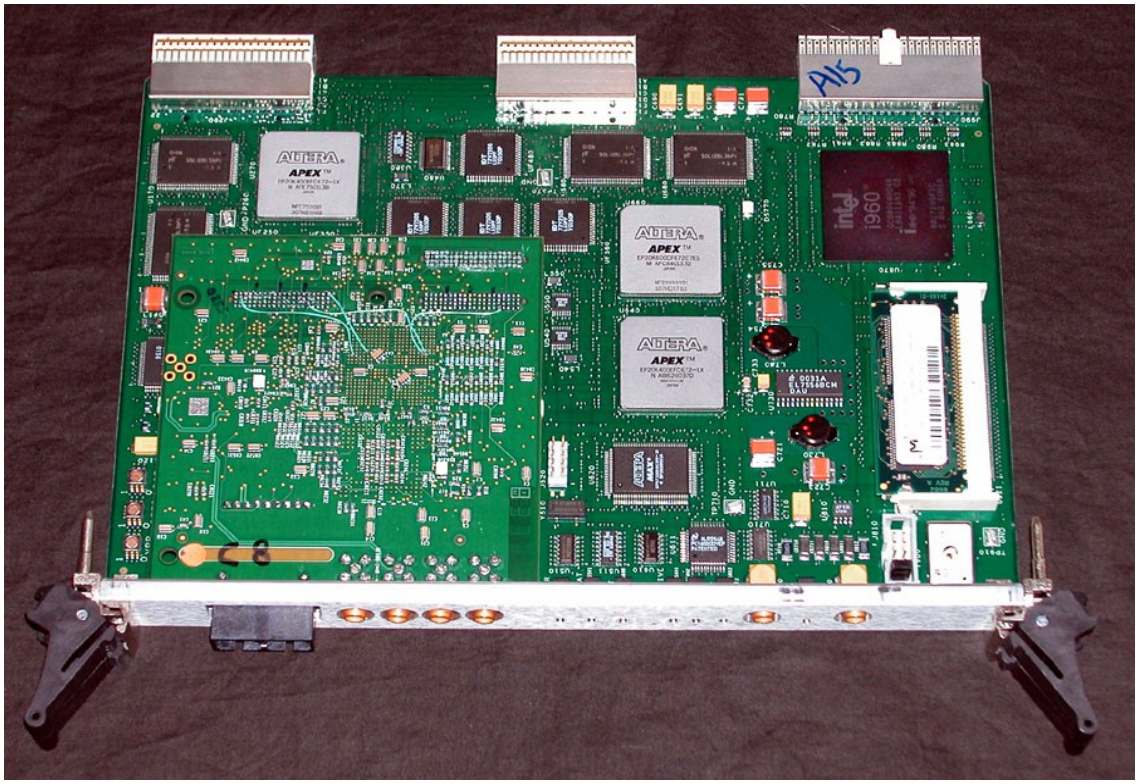


Figure 15 Photograph of a UNAE engine board

To take advantage of the opportunity to reuse previously developed video interfaces, the UNAE base card has mezzanine connectors sufficient to host either a single network interface mezzanine card or a pair of video mezzanine cards, as described in section 0, Mezzanine boards.

The UNAS system architecture originally required the UNAE to support a network interface mezzanine with a single BRAD ASIC to perform both multirate clock and data recovery (CDR) and serialization/deserialization (SERDES) functions. The BRAD was a key element to the application flexibility of the UNAS. Due to long lead times in designing and fabricating the BRAD, an optical test mezzanine was designed and fabricated, using a newly-available commercial chip that provided CDR and SERDES functions. Because the UNAE was expected to interface with the BRAD, it was designed to distribute a buffered version of the same clock to each of the engine's FPGA's. This meant that later, in order to provide the two clock rates necessary to support Gigabit Ethernet using the same hardware without support from the BRAD, some modification of clock distribution would be necessary. As described later, these modifications were all accommodated in a modified mezzanine, leaving the UNAE design intact.

The UNAE is realized with a twelve-layer circuit board, the first run of which was fabricated in June of 2000 and the second run of which was fabricated, without changes, in January of 2002. Design elements of the UNAE board are presented below.

4.3.2.2 Hardware functions

4.3.2.2.1.1 I/O mezzanine interface

Referring to Figure 4, the UNAE transmit and receive serial interface is implemented as an interchangeable mezzanine. Three connectors provide hosting for 32 bits of LVTTL transmit and 32 bits of LVTTL receive data. Two independent and fully programmable (20-100Mhz) reference clocks are provided to the mezzanine, and two word synchronization clocks are expected from the mezzanine for each data direction. The mezzanine is the master for both transmit and receive word clocks. This is consistent with modern communication IC's, including the HDTV interface chips used on the HDTV mezzanine.

There are general purpose IO pins assigned to the mezzanine interface that are under software control via the Framer FPGA. They provide control and status signaling to configure and monitor the mezzanine.

There is also a common connector (CC) bus that provides eight bits of bussed data between mezzanine connectors C1 and C2 and the Framer FPGA. This provides legacy mezzanine type reads.

Each mezzanine type presents an identity discovery mechanism so the SW and HW together can configure appropriately. SW sets all pins as INPUTs and runs a detection algorithm for the expected mezzanine. If it fails, then the reset of the setup is aborted. If it succeeds it configures the appropriate INPUTs and OUTPUTs. It is possible to build a mezzanine board auto-detect feature in software. However, this has not been implemented since it requires software policy enforcement to overcome system wide configuration issues.

4.3.2.2.1.2 Interface processor

The UNAE board includes an Intel i960RD interface processor. The embedded processor running VxWorks real-time OS provides support functions for the ASIC and FPGA functional blocks, in addition to managing protocol selection decisions and handling the loading of FPGA images specific to supported protocols.

It communicates with the functional blocks on the UNAE card through registers and a messaging passing protocol. Message passing is utilized as described in the Software Architecture specification for communication with the external host and GUI.

The interface processor manages the primary PCI backplane interface, which carries commands to and from the host processor on another module. The interface processor core circuit was duplicated from a previous Tektronix-designed Trillium MPEG Video Encoder board. The design includes the VxWorks operating system, a PCI device driver and a messaging unit. The following changes were made to the Trillium design to enable integration into the UNAE board:

- Added FPGA configuration signals to Load PLD to allow independent, processor-controlled programming of three FPGA devices.

- Updated secondary PCI bus interface design to allow multiple PCI devices. Brought out two four-bit signals, S_GNT<4..1>* and S_REQ<4..1>*.
- Brought out local bus address, data, and control signals.
- Moved Load PLD to output side of decoder and data/address drivers. Changed local bus interface to Load PLD to separate data/address buses.
- Added UART.
- Added diagnostic connectors.

4.3.2.2.1.3 Loader

A non-volatile programmable logic device (PLD) is used to control configuration of the three FPGA's, as well as reading of the board type and version registers. The processor writes FPGA configuration data to the Load PLD, which in turn strobes the hardware signals appropriately to program the FPGA's. The Load PLD is programmed after manufacturing through its JTAG port.

4.3.2.2.1.4 Framer FPGA

The Framer provides the first interface to/from the mezzanine data path, the mezzanine control path, access to the three registers ports of the DDS, and LED control. There is a PCI interface through which it is connected to the embedded processor.

The intended function of the Framer is to extract/insert serial line overhead at the mezzanine side on deserialized data, and transmit/receive parallel, protocol-encapsulated payloads to/from the PLPE. There are four clocks provided to the Framer: The Buffered Write Word Clock, the Buffered Read Word Clock, the PCI clock, and the 10Mhz GPS clock. All Framer FPGA images have the DDS, mezzanine, and LED control registers mapped at the identical address.

The Framer can communicate with the mezzanines bi-directionally using registers and the secondary PCI interface. Parallel reads and writes with the optical network mezzanine (g2469xc version) can also run through this PCI interface.

Framer designs for specific protocols are described in sections 4.4 and 4.8 of this report, and in additional specifications ("SONET Framer Preliminary Specification" and "Gigabit Ethernet GMII Bridge Specification NGI UNAE Framer Programmable Logic Device") developed within the program.

4.3.2.2.1.5 PLPE FPGA

The PLPE (physical layer processing engine) accepts or sends frame-stripped data (e.g. HDLC data and or SMPTE-292 video data) from or to the Framer and extracts payload. Alternatively, it functions as the MAC processor for Gigabit Ethernet. There is a PCI interface through which it is connected to the embedded processor. Details of the roles it plays depend on the protocol data format being processed. Its roles in the implemented formats are included in other sections of this document and in respective specifications developed in the program.

For formats that require recovery of clocks for real time user data, the PLPE and CPE work interactively to fill and empty the intervening FIFO of data in a manner that is determined by arrival rate of the user data. This may be supplanted by clock control provided by a GPS receiver.

4.3.2.2.1.6 CPE FPGA

The cell packet engine acts first and foremost as a simple, rate-matching and address filtering interface controller between a pair of hardware FIFO's and the SPI-3 up/down bus backplane to manage cell flow. On the receive side, the engine will extract cells from the receive-side FIFO and hand them to the link-layer device. On the transmit side, the engine will pass cells from the link-layer device to the transmit-side FIFO. In all cases, the engine must monitor the fullness of the FIFO's and create the appropriate request/grant signals.

Features of the CPE (not all implemented) include

- Programmable logic to enable multiple applications
- Can support various width and rate interfaces (SPI-3 implemented)
- Manages packet FIFO's
- Programmable to isolate/drop streams through address filtering
- RAM-based counters enable QOS measurements
- Large packet buffers permit out-of-order detection/correction
- Supports packet insertion and extraction
- Traffic generator creates streams (not implemented)
- Captures packets for subsequent analysis (not implemented)

There is a PCI interface through which it is connected to the embedded processor. Two external 36x128K scratchpad memories are available to store intermediate data. These resources are configured via program images in the CPE to create engines to perform tasks on the cell/packet stream.

The scratchpad memory has been used to buffer and manage the packet flow. For example, by buffering the arriving packets, out-of-order packet arrival can be corrected up to the limit of the buffer size. The scratchpad can also be used as a capture buffer to retain packets from a set of flows for later analysis.

4.3.2.2.1.7 On board memory

On board DRAM is made available for VxWorks core operating system, which DMA's the UNAE FPGA images into the RAM at UNAE boot time. The VxWorks core operating system will then dynamically link the software module to the core OS. The board is designed to accommodate up to 32 MB of DRAM.

There is a 1 MB FLASH memory requiring 12 V to program, applied through a hardware switch for write protection. This Flash memory contains the core VxWorks operating system. There is not any UNAS system information contained in this image. The image is placed into the FLASH ROM by use of the VxWorks Loader. The FLASH memory is

initialized once at board turn on with a sufficient VxWorks image to initiate a boot of the UNAE core OS upon system start or reset. Upon this initial boot, VxWorks then writes a full system image into DRAM, allowing it to take full control of more completely initiating the full board to a known stable state.

4.3.2.2.1.8 Backplane bus interface

Each cPCI 6U size UNAE card fits into the Tektronix Platinum platform, a cPCI based platform for communication test equipment, which has a variety of chassis options, from 5 card slots through 16 slots. There exist four 32-bit interfaces on the Platinum Platform backplane. These are termed an up/down bus structure. In addition to the data lines, there are enough additional auxiliary signals to complete two SPI-3 interfaces.

As shown in Figure 16, one interface is configured as a Link layer interface and the other is the physical (PHY) layer interface.

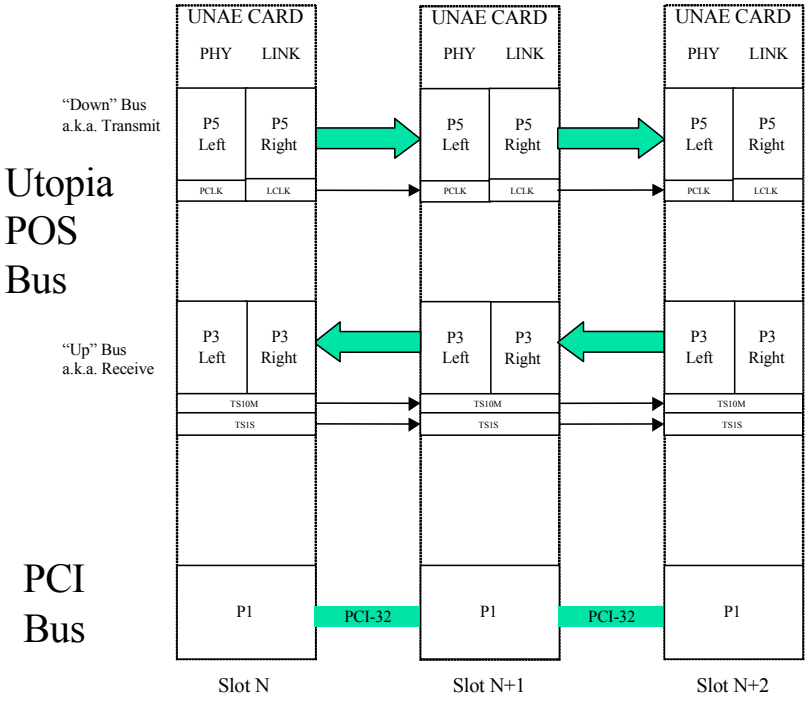


Figure 16 Busing among multiple UNAE cards in a Platinum platform

For a card in slot n, its Link layer partner is in slot n-1 and its physical layer partner in slot n+1. The Link card will be the source of the backplane clock. The Link configuration is typically applied to UNAE cards used for higher layer protocols such as processing video data into and out of packets. The PHY card typically acts as a network physical layer interface.

At the system level, if there is a NIC and a user interface card in use, the transmit and receive directions are referred to the network interface (i.e. transmit is to the network).

When discussing UNAE components, transmit and receive refer to the function of the component. For example, there are both transmit and receive FIFO's (TFIFO and RFIFO, respectively) in the transmit path. The TFIFO and RFIFO refer to the data flow with respect to the backplane SPI-3 bus.

There is not support for multiport PHY due to a limit of available addressing lines, nor is there support for extensions such as in-band addressing for the transmit interface. The CPE FPGA has the responsibility of acting as the up/down bus controller for the UNAE as described below.

Two UNAE cards are interconnected across the backplane via the CPE FPGA. Figure 16 shows how the SPI-3 busses on a UNAE card in slot n are connected with adjacent cards in slots n-1, n+1. Connector P5 is the transmit data path for the physical interface. The link end is Link Transmit (LT). The physical end is Physical Transmit (PT). Connector P3 is the receive data path for the physical interface. The link end is Link Receive (LR). The physical end is Physical Receive (PR).

4.3.2.2.1.9 FIFO

Located between the CPE and PLPE FPGA's, synchronous FIFO's are used to buffer data between the PLPE and the backplane. The FIFO is a standard off-the shelf component. Each FIFO is 256k x 36 bits wide and carries 32 bits of data and a four bit TAG field. The TAG field provides interpretation of the contents of the data field and packet structure. The lower two bits of the TAG field are compatible with the TAG field of the SPI-3 interface as shown in the accompanying table below.

While these TAG bits can identify the valid bytes in the data word, they are not required. For IP packets, the header will be identified by the start of packet tag. The length sub-field will be located and the packet length counted down. When the count has been exhausted, the FIFO will be "flushed" and the data discarded until the next start-of-packet signal is asserted.

Table 2 FIFO packet tags indicating packet data type

Tag	Meaning
0000	Normal word; data[31:0] valid.
0001	Reserved
0010	Reserved.
0011	Timestamp[31:0].
0100	End of packet; data[31:0] valid.
0101	End of packet; data[31:8] valid.
0110	End of packet; data[31:16] valid;
0111	End of packet; data[31:24] valid.

1000	Start of packet; data[31:0] valid.
1001	Start of packet; data[7:0] valid.
1010	Start of packet; data[15:0] valid.
1011	Start of packet; data[23:0] valid.
1100	Discard Packet
1101	Simultaneous start/end; data[31:8]/data[7:0] packet boundary.
1110	Simultaneous start/end; data[31:16]/data[15:0] packet boundary.
1111	Simultaneous start/end; data[31:24]/data[23:0] packet boundary.

The TFIFO only goes above zero occupancy when the backplane bus is congested.

The RFIFO normally runs at half-full under the control of the clock recovery algorithm and subject to cell or packet interarrival time variations.

As it is not known a priori whether the transmit or receive side of the FIFO should be responsible for programming these parameters, both the PLPE and the CPE will be connected to the programming pins. Later, software will determine which device should do the programming. At that time, the buffers on the programming EPLD will be enabled and programming performed.

In addition to the direct FIFO connections, there will also be reserved signals routed between the PLPE and the CPE. These lines are defined as needed to perform functions such as notification of a complete cell added or removed, communications relating to fill level, and so on.

4.3.2.2.1.10 UART

A UART was added to enable a terminal interface directly to the operating system running on the i960RD without passing data through the backplane PCI bus. The UART implementation was made to resemble that of the PCI controller board from Cyclone Microsystems motherboard as closely as possible, including address map and hookup to the XINT7 interrupt.

The UART is clocked using a one-time-programmable crystal oscillator at 1.8432 MHz. The oscillator functions from power-up.

The Data Terminal Equipment signals are buffered with 14C88/14C89 RS-232 receiver and transmitter chips. They are available on a 10-pin header, which can be connected by a ribbon cable to a 9-pin D-SUB connector.

4.3.2.2.1.11 SRAM interface

The architecture includes SRAM connected to the CPE and PLPE blocks to allow for

buffering of bursty packet flows, reordering of out of order packets or cells and if needed can be used for more flexibly packetizing and reassembling data as described in the previous section 4.3.2.2.1.6 on the CPE FPGA.

There are two independent scratchpad memories attached to the CPE. These RAM's are clocked with the same clock as the CPE. There is a two clock period latency for any read or write operation, but reads and writes may be overlapped as the part is pipelined. The two RAM's are available for whatever need should arise. Both the CPE and PLPE have RAM interfaces, although only the CPE has been implemented.

4.3.2.2.1.12 Loopback functions

Each FPGA has the ability to loopback the incoming data from its neighbor sourcing the data. This is used for debug and has been one of the most helpful features in many respects. It is implemented by adding code to each FPGA and can therefore be very flexible in its implementation.

4.3.2.2.1.13 DDS Clock generation

There are three Direct Digital Synthesizer (DDS) clocks. Transmit clock, backplane clock and FIFO readout clock are each controlled by a DDS. When generating output for a protocol where the output rate must match the input rate, the synthesizer monitors the fill level on the rate-matching FIFO and makes incremental adjustments to maintain the proper FIFO fill level. Only video such as SMPTE-292 and SMPTE-259 have this requirement. When generating outputs where exact rate matching is not a requirement, the synthesizer generates a fixed frequency clock. DDS devices are controlled by the Framer.

The receive portion of the UNAE board includes dedicated circuitry to recover a video clock frequency that is equal to that of the remote transmitter when integrated over the long term. The clock recovery circuit is a software controlled clock generator with an extremely precise "vernier" for adjusting the frequency. It is based on the Analog Devices AD7008 DDS chip in conjunction with an ICS AV9170 clock multiplier. The circuit design used is a patented Tektronix invention.

4.3.2.2.1.14 GPS Timing Reference

The front panel optionally has connectors to accept a 10 MHz reference clock and a 1 Hz reference pulse from a Global Positioning Satellite receiver. The software, via a COM port connection will receive timing data from the GPS and synchronize timestamp counters to UTC. The GPS signal may be made available to other cards via two pair of full-differential ECL signal lines bussed on the backplane to all slots. These drives must be capable of driving 25 ohm lines and have the ability to be disconnected electrically from the backplane should another card be selected to drive the clock lines.

The GPS timing hardware is not fully implemented and has not been tested, but the packet header for transport of SMPTE292 video includes timestamps if provided.

4.3.2.2.1.15 Reference Oscillator

A local 10 MHz oscillator provides a timing source for the DDS subcircuits, timestamp clock, and, via a clock multiplier, the 80 MHz clock for the backplane interfaces. When available, this reference may be switched out through use of multiplexor in favor of the 10 MHz reference from the GPS timing module.

4.3.2.3 Physical Interfaces

4.3.2.3.1.1 Backplane interfaces

As described above, the UNAE meets P3/P5 connectors of the Platform up/down bus, which is compliant with UTOPIA3 ATM interface and also with the SPI-3 packet interface, without address selection. The cards also meet with cPCI 64 bit bus on P1/P2 for communication with the host processor on a separate module.

4.3.2.3.1.2 RS232 Connector

A 10-pin ribbon cable header is provided for RS-232 serial port connection to the i960RD processor core. The signals are arranged on a 9 pin D-SUB style connector when connected via a ribbon cable. Refer to the RS-232 standard for signal name definition.

4.3.2.3.1.3 JTAG Header

A JTAG header is provided for direct connection to JTAG test instruments. The pinout is specific to the JTAG test instruments previously used in Tektronix internal manufacturing.

The JTAG header may be shorted out of the JTAG loop by placing a zero-ohm resistor. In this case, the JTAG connection is made through the primary PCI bus.

4.3.2.4 Software/Hardware Interface

4.3.2.4.1.1 I960 Memory Mapping

The board-level chip enable is asserted when a memory access is requested by the processor or PCI Address Translation Unit (ATU) into the Memory Bank 1 control space. This space is defined by setting Memory Mapped configuration Registers upon startup. The Memory Bank 1 base addresses and offsets are cataloged in the “UNAE Hardware/Software Specification Document.”

4.3.2.5 Secondary PCI Bus

The local bus is used for writing FPGA configuration registers. The three FPGA’s are accessed through the secondary PCI bus.

4.3.2.6 Fabrication

For the UNAE board a twelve-layer circuit board was captured in Cadence Concept, final net and pin definitions for all components were completed, FPGA pin assignments were

compiled and timing was found to be acceptable. A design review, routing of the circuit board, and bill-of-materials was completed in July 2000 and the board was released for fabrication.

A multirate optical test mezzanine board was architected, and schematic capture completed. Long lead time components were ordered and all components scheduled to arrive in the first week of September 2000. The board layout was released for fabrication; and boards scheduled to arrive in September 2000. Just these two fabrication runs of the single G2468-XA UNAE board design were necessary to complete the program. Sixteen boards were fabricated of which fourteen were functional at end of program.

4.3.2.7 Turn on, debug and performance of UNAE boards

Two fabricated boards were received from a short run, with the balance to be built in September/October 2000. The two fabricated boards passed initial power up and were installed in the platform, where VxWorks was successfully loaded in the embedded processor. Images for all three FPGA were successfully loaded and partially debugged.

The PCI interface embedded in each FPGA was brought to full functionality in the last quarter of 2000, while several minor iterations of the programmable logic designs for each FPGA part were needed to get basic functional performance. PCI accessible interface logic was added to the Framer to control the (direct digital synthesizer) DDS's and mezzanine. A general purpose I/O interface, fully programmable at the individual pin level was created. Embedded test software was written to identify each of the mezzanine types (HDIN, HDOUT or optical test mezzanine) and to configure the detected mezzanine, FPGA's, and DDS frequencies for proper operation.

Two separate sets of FPGA code were initially developed for the three FPGAs on the UNAE boards. The first set was designed to accept SONET-framed IP data from an optical network, and interface the packet data to the SPI-3 interface. The second set of FPGA code was installed on a second board to achieve a demonstration of HDTV transport in packets over a SONET network. Details of both designs are provided below.

Besides debugging the electrical integrity, speed and timing of individual components, substantial FPGA code modifications were required at each progressive step both to confirm functionality and especially to debug in-circuit performance. Flexibility offered by FPGA's and the ability to modify designs helped eliminate signal integrity and timing issues of the UNAE boards.

The first functional code for the optical network interface was completed and simulated in June, 2000. The code consists of the following components:

- Framer: takes SONET data at an OC-48 rate (2.5 Gbps) from the optical layer, removes the SONET overhead, aligns the data, and presents it on a 32-bit wide interface. Code for controlling DDS chips and mezzanine cards is also on this part.

- POS48 PLPE: takes parallel data from the Framer, removes HDLC/PPP overhead, and passes the data to a 36-bit wide hardware FIFO. This FPGA passes data to / accepts data from the microprocessor in order to control setting up a PPP link.
- CPE: takes data from the 36-bit wide FIFO and creates the appropriate control signals before passing it to the SPI-3 bus.

The UNAE board was tested on the bench using PPP traffic generated by an IXIA OC-48 test set. It was possible to set up a PPP session through the microprocessor interface, and have the UNAE card receive error-free test packets from the IXIA box (based on the PPP frame check sequence calculation over all of the data).

FPGA code for video functionality on the UNAE board was also written and debugged consisting of the following components:

- Framer: takes a parallel high definition video stream from an HD mezzanine card (aggregate data rate 1.5 Gbps) and passes it through to a 20-bit wide interface. Code for controlling DDS chips and mezzanine cards is also on this part.
- Video PLPE: takes 20-bit wide parallel video data from the Framer, segments the video for transmission in IP packets, adds RTP, UDP, and IP headers to the data, and passes it to the hardware FIFO. This chip also checks for the type of HD video present, and monitors video errors using the SMPTE-292M CRC.
- CPE: takes data from the 36-bit wide FIFO and creates the appropriate control signals before passing it to the SPI-3 bus. The code is identical to that placed in the optical network interface described above.

Board to board communications were not successful, due in part to incompatible SPI-3 bus clock rates and board components. After getting faster FIFO's installed and changing the backplane DDS clock rate to 80Mhz, the video loopback through the CPE began working, demonstrating that HDTV to IP and back could be performed on the UNAE. Performance of the video card was tested by sending serial HD video test patterns generated by a Tektronix TG700 test set to the board. Video serialization/deserialization was done on mezzanine cards attached to the UNAE board that were developed for a previous project. Details of these results are provided in section 5.2.

By February 2001, eight UNAE boards had been manufactured and modified for operation at OC-48 line rates. Seven of these boards operated error-free in loopback mode with serial HD video signals. Error-free loopback of OC-48 SONET signals with SONET/PPP code loaded into the UNAE FPGAs had not yet been demonstrated. The UNAE had been able to send and receive PPP configuration packets in order to establish a PPP link with a Cisco 12000 GSR router equipped with OC-48 line cards. There were still timing issues with the main data path. Finally, the PLPE FPGA was replaced with a faster, just-released part.

By September 2001 error free video transmission was demonstrated between two UNAE boxes attached to different OC-48 ports of a Cisco router, demonstrating that the UNAE concept was viable and signal fidelity was sufficient for this major milestone. Subsequently, this was demonstrated to work on a WAN from Seattle to Arlington, VA

using Internet2 as described in detail in section 5.2.1.

The UNAE board was also utilized to provide interfaces between SDI video and Gigabit Ethernet and between IP packet generators and all-optical label switching network elements. These applications are detailed below in sections 5.2 and 5.3.

4.4 SONET OC-48 Framer design

4.4.1 SONET Framer design objectives

The SONET framer implements all the functions necessary to create and recover SDH/SONET STM-16c/STS-48c frames. Separate transmit and receive designs are implemented in a 400k gate FPGA using the Protocol Compiler tool. Input and output paths are 32 bits wide, with several additional control lines. On the receive side a data-valid flag indicates when SONET overhead is being processed; on the transmit side, a request line provides an indication several clocks ahead to the upstream device to hold off data transmission while SONET framing and overhead is generated. A PCI interface was also implemented in the device using Protocol Compiler for the CPU to access registers in the design for control and data collection purposes.

4.4.2 Summary of work

4.4.2.1 SONET transmit design

On the transmit side, an SDH/SONET STM16c/STS-48c frame is constructed and the appropriate overhead bytes synthesized and placed in their proper locations. Some selected key overhead bytes, which are not determinable a priori, are inserted from registers. At the proper times, the contents of the Synchronous Payload Envelope (SPE) are taken from yet another 32-bit interface and placed into the payload. B1, B2, and B3 parity calculations are performed and the results placed into the frame. Provisions have been made to programmatically insert either path or line Alarm Indication Signal (AIS) if desired. Finally, the frame is scrambled and passed on to the final 32-bit interface for multiplexing into the serial stream.

4.4.2.2 SONET receive design

The receive side of the framer begins by accepting an unaligned 32 bit word version of the serial stream. It searches the incoming data until it locates the framing pattern. Based on the location of the framing pattern, the word is shifted to align the frame. Once synchronized, descrambling is performed and key bytes located. The pointer is extracted and the path overhead location determined. All pointer movements are completely supported as per the applicable standards. Various key section, line, and path overhead bytes are made available via capture registers. Along the way, B1, B2, and B3 parity bytes are recalculated and compared to the values specified in the frame. Three separate 32-bit counters keep track of the number of errors. The SPE is delimited and the payload data is made available via a 32-bit interface.

4.4.2.3 SONET PCI interface

The Framer FPGA passes information to and from the Intel i960 embedded microprocessor on the UNAE board through a secondary PCI interface. This secondary PCI interface is invisible from the Windows NT side (backplane PCI interface) and is used only for communications with the FPGAs on the board. Communication with the UNAE mezzanine boards via the secondary PCI interface is also possible through the Framer FPGA. Two Framer registers, c1[19:0] and c2[18:0], allow 39 bits of information to be written to or read from any of the mezzanines. In addition, the latest version of the optical interface mezzanine (g2469xc) has a data bus cc_data[7:0] and an address bus cc_addr[7:0] that allow parallel reads and writes from the mezzanine via the Framer PCI interface.

The Framer PCI design is a target-only PCI interface (i.e., cannot act as bus master) written in the Protocol Compiler graphical programming language. The design requires 5 clocks for a PCI transaction (read or write) because additional wait states were added to the design to allow time for data propagation to and from the mezzanine cards. The PCI design uses address bits adr[26:24] to determine the target FPGA; for the Framer these bits are set to "001". Addressing for the registers in the Framer is built into the PCI design block (see the SONET Framer Specification for addresses and descriptions of the registers). All writeable registers are read-back except for the mezzanine registers c1[19:0] and c2[18:0].

4.4.2.4 SONET design verification

Verification of the SONET Framer design was done in two stages. For preliminary design testing and debugging of the SONET encoder, a Protocol Compiler test bench was written that generates a sequence of small packets and places them at the Framer input. The SONET output of the design is checked by inspection of waveforms. Testing of the SONET decoder design is difficult by this method because the large amount of data required to generate complete OC-48 SONET frames (38,880 bytes) makes testing very tedious.

For complete testing of the SONET Framer design using random data, a test bench was written using the VERA language. VERA is a superset of the Verilog hardware design language with extensions to simplify verification and debugging of designs. A VERA test bench was written that feeds the output of the SONET encoder into the SONET decoder and checks the resulting output. The SONET data path was verified to be free of errors; however, the SONET loss-of-frame error indication does not work properly.

4.5 Packet Over SONET OC-48 PLPE design

4.5.1 Packet Over PLPE design objectives

This design, shown schematically in Figure 17, implements the Packet-over-SONET (POS) protocol in the physical layer protocol engine (PLPE) FPGA of the Universal Network Access Engine. This FPGA is the interface between the Framer FPGA and a

hardware FIFO on the UNAE board. The FPGA will extract (receive direction) or encode (transmit direction) IP packets in PPP/HDLC framing. The design was implemented using the Protocol Compiler graphical hardware design language.

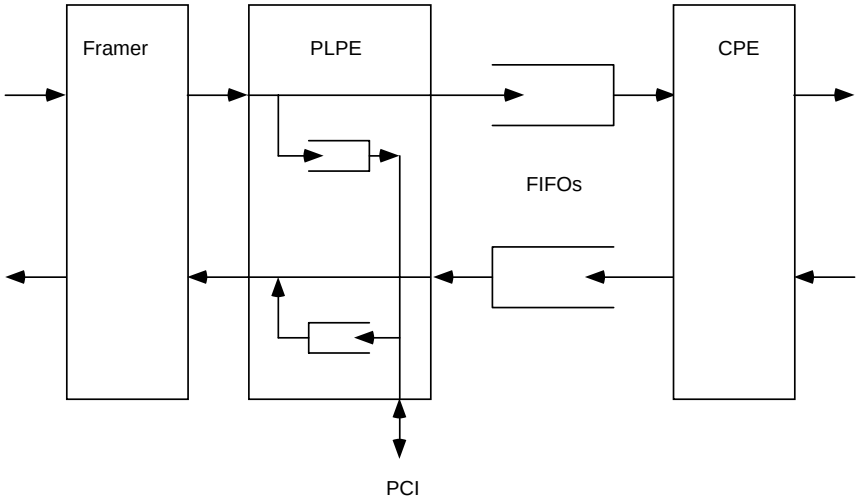


Figure 17 . Placement of the POS-PLPE FPGA within the Universal Network Access System architecture. I/Os on the left side are PPP/HDLC streams in SONET, and I/Os on the right side are IP packets

4.5.2 Summary of work

4.5.2.1 POS transmit design

In the transmit direction the POS-PLPE FPGA will read data from a FIFO, encapsulate the packets in PPP/HDLC headers, and pass the data to the SONET Framer FPGA. The data flow will be controlled by a request line from the Framer. During the PPP setup stage the POS-PLPE FPGA will also accept packets from the microprocessor interface and encapsulate them as above.

4.5.2.1.1 Transmit direction processing

1) Data input:

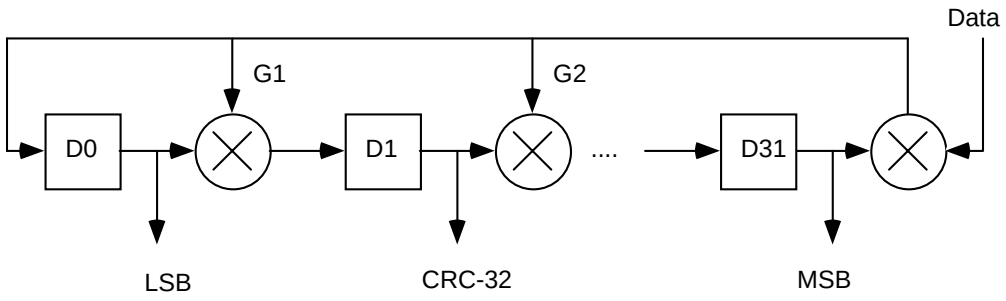
Data is read from the microprocessor interface FIFO during PPP setup, or from the main data FIFO during normal operation. Availability of a packet from the microprocessor for transmission is indicated by setting bit TRN_CTL[4]. Packets from the microprocessor will have the following PPP/HDLC frame generation steps already done by software, and will be passed through unchanged.

2) Frame generation:

Frame generation is done by inserting Flag Sequence characters (0x7e) at the start of packets, followed by the HDLC Address and Control bytes (0xff03). Following this the 2 byte PPP Protocol field is inserted, set to either 0xc021 or 0x8021 for data from the microprocessor interface or to 0x0021 for data from the main data FIFO interface (this field can optionally be set to an alternate value via the TRN_PROTO register). If no packets are waiting in the FIFOs to transmit, Flag Sequence characters are transmitted. In the event that the FIFOs become empty during transmission of a packet, or if the errored packet signal is asserted during transmission, the abort packet sequence 0x7d7e is inserted.

3) FCS generation:

Each data word is processed as it arrives to calculate the CRC-32 Frame Check Sequence (see diagram below). The 1's complement of the 4 byte FCS is appended to the last data byte of the frame.



4) Byte stuffing:

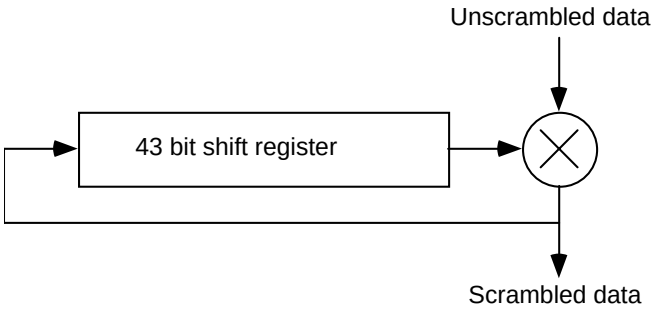
The data words are processed to determine if any of the bytes contain the Flag Sequence (0x7e) or Control Escape (0x7d) characters. If they are found, they are replaced by the the 2-byte sequences 0x7d5e and 0x7d5d, respectively

5) Frame termination:

Flag Sequence characters (0x7e) are appended to the end of the HDLC frame so that it ends at a 32-bit boundary. If data already ends at a 32-bit boundary, an additional four Flag Sequence characters are added.

6) Data scrambling (optional):

Scrambling is done on the entire POS frame sequence, including the FCS and framing flags. Scrambling is done using a parallel self-synchronous $X^{43} + 1$ scrambler as shown below.



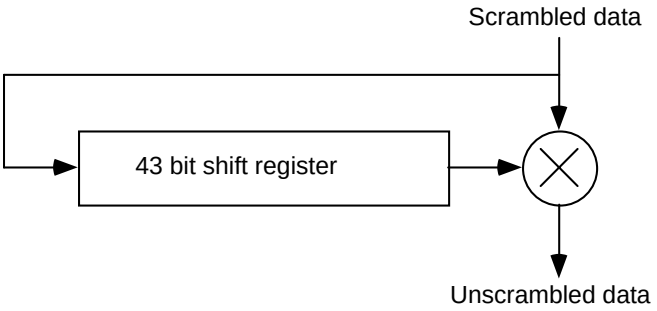
4.5.2.2 POS receive design

The POS implementation of the PLPE in the receive direction takes data from the SONET Framer FPGA, processes packets encapsulated in HDLC/PPP headers, and sends the packets to an external hardware FIFO or an internal FIFO for PPP LCP negotiation packets. Figure 17 above indicates schematically the placement of the PLPE functionality in the Universal Network Access System. The interface to the POS FPGA from the SONET Framer is a parallel data bus plus a valid line that indicates whether each clock contains valid data.

4.5.2.2.1 Receive direction processing

1) Descrambling (optional):

Descrambling is done using a self-synchronous $X^{43} + 1$ descrambler as depicted below:



2) HDLC frame delineation:

Frame delineation is done by searching for "Frame Sequence" patterns (0x7e) in the descrambled data. Single or multiple Frame Sequence patterns are used to mark the beginning and end of HDLC frames; repeating Frame Sequence patterns are used as byte-aligned fill. The Frame Sequence patterns are removed from the data and it is passed to the next process.

3) Byte “destuffing” (optional) and invalid frame detection:

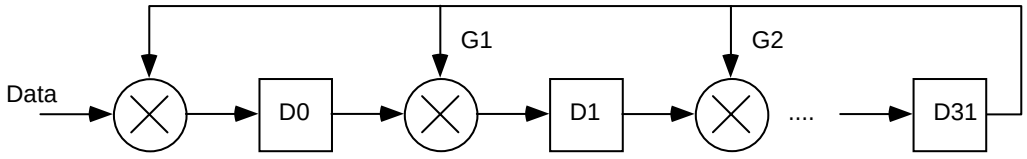
Certain characters (0x7e and 0x7d) are “escaped” in the PPP data stream by modifying them and placing the Control Escape character (0x7d) in front of them. The data is parsed for Control Escape characters, and when they are found they are removed and the following character XORed with 0x20, except if the following character is 0x7e. This sequence signifies an aborted frame, so the “errored data” flag is set and the aborted frame counter is incremented.

4) Parse PPP protocol and length fields and forward packet to the appropriate interface:

The first characters of an HDLC frame are the Address and Control bytes, which are always set to 0xff and 0x03 for PPP. These bytes are discarded for IP data packets (passed through unchanged for control packets routed to the microprocessor), and the following two bytes (PPP Protocol field) are examined. If the LSB of the least significant octet is not 1 and the LSB of the most significant octet is not 0, the frame is an unrecognized protocol and is marked for discard. If the first two bytes are 0xc021 (Link Control Protocol) or 0x8021 (IP Control Protocol), the frame is forwarded to the microprocessor interface FIFO and a status bit is set (maskable interrupt) indicating that data should be processed by the processor. Frames with the Protocol field set to 0x0021 are forwarded to the main data FIFO interface.

5) FCS check, min and max packet length check (optional):

After byte destuffing the data is processed word by word to calculate a CRC-32 Frame Check Sequence (see diagram below). The generator polynomial for CRC-32 is $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$, where each non-zero exponent corresponds to a feedback path labeled G1, G2… in the diagram below. The register D1…D31 is loaded with 1’s at the start of reception. If the CRC-32 including the embedded FCS (last four bytes of the frame) are not equal to 0xdebb20e3, the data is errored. The FCS error counter is incremented and the frame is marked for discard.



A count is also kept of the number of bytes in the frame. If this number is below the minimum set in the MIN_RU register, the min error counter is incremented and the frame is marked for discard. If the number is above the maximum set in the MAX_RU register the max error counter is incremented and the frame is marked for discard.

6) Receive timestamp:

Each data word forwarded to the receive FIFO is timestamped with a 36-bit quantity

derived from a highly accurate 10 MHz clock (on board oscillator or GPS clock in an adjacent PCI slot). 32 bits are a 10 MHz counter, and 4 bits are a sequence number added to each 32-bit timestamp so that all timestamps of data at the 77.76 MHz rate are unique. The timestamp is forwarded out a separate, parallel FIFO port at the same time as the data.

4.5.2.3 POS PCI interface

The Packet-over-SONET PLPE PCI interface is similar to the Framer PCI interface. The design is a target-only PCI interface (i.e., cannot act as bus master) written in the Protocol Compiler graphical programming language. The design requires 5 clocks for a PCI transaction (read or write) because additional wait states added to the Framer PCI design were left in the PLPE PCI design to assure reliability. The PCI design uses address bits adr[26:24] to determine the target FPGA; for the PLPE these bits are set to "010". Addressing for the registers in the PLPE is built into the PCI design block (see the Packet-over-SONET PLPE Specification for addresses and descriptions of the registers).

4.5.2.4 POS design verification

The Packet-over-SONET designs were also tested in two stages. During initial design, short input sequences were generated using a Protocol Compiler test bench that allowed for quick debugging of the transmit and receive designs. More exhaustive testing using long sequences of random data was then done using test benches written in the VERA language. Automated error-checking allowed verification with tens of thousands of test packets of varying length.

4.5.2.5 PPP negotiation

PPP connection negotiation will be handled by a microprocessor on the UNAE board. PPP packets whose protocol field indicates that they are control packets (Link Control Protocol or IP Control Protocol) will be placed in a microprocessor FIFO implemented on the POS-PLPE FPGA. When the channel negotiation is completed, the microprocessor will signal the POS-PLPE via a control bit and data packets will be forwarded to the main hardware FIFO interface. A 36 bit timestamp is added to each data word and written into a parallel hardware FIFO coincident with the data.

The transmit function uses the following default PPP parameters:

Packet length: 572 bytes (settable 48 – 65,536 bytes)

Asynchronous control characters: not supported

Authentication protocol: authentication not required

Quality protocol: link quality monitoring disabled

Magic number: magic number negotiated by software

Protocol field compression: not supported (2 byte protocol field)

Address and control field compression: not supported (2 bytes address and control)

FCS alternatives: CRC-32 supported in hardware, CRC-16 implemented only for negotiation packets

4.6 Serial digital video PLPE design

4.6.1 Video PLPE design objectives

The video implementation in the PLPE FPGA takes video data from a video mezzanine card (forwarded through the Framer FPGA) and inserts it into IP/UDP/RTP packets in the VIN direction, or removes it from packets in the VOUT direction. The data width is 10 bits for the standard definition SMPTE 259M implementation, and 20 bits for the high definition SMPTE 292M implementation. The clock rate for the SMPTE 259M implementation is 27 MHz (270 Mb/s data throughput) and the clock rate for the SMPTE 292M implementation is 74.25 MHz (1.485 Gb/s data throughput). The Video PLPE is intended to allow transmission and reception of packetized data with an adjacent physical layer network card.

In the video input (VIN) direction, the Video PLPE will read 10 or 20 bit video words, format them into 32 bit data words, construct packet headers from information in the FPGA registers, and forward the packets to a hardware FIFO (RFIFO). From there the packets will be sent to a neighboring card across an SPI-3 backplane interface. Since video input is a continuous stream, the backplane interface needs to be run at a high enough speed to accommodate the video data rate plus the additional overhead of packet headers. We have chosen to run the backplane interface at 80 MHz, allowing for a 72% packet overhead when HD video data is transmitted. This requires that a minimum length packet have at least 205 bits of video data for a header length of 44 bytes (20 bytes IP + 8 bytes UDP + 16 bytes RTP). Since we require that a packet contain an integral number of 10-bit video words and start and end on 32-bit boundaries, the minimum packet contains 40 bytes or 32 video words.

In the video output (VOUT) direction, packetized video is removed from a hardware FIFO (TFIFO), packet headers are removed, and a 10 or 20 bit wide parallel video stream is reconstructed to be forwarded through the Framer to a video output mezzanine card. The output data rate is matched to the input data rate by buffering data in the FIFO and controlling the output data clock from an average FIFO occupancy. With appropriate software control, this will reconstruct the original video rate exactly as long as no data is lost. Sequence number checking has been implemented in the design to determine whether any data has been lost.

4.6.2 Summary of work

4.6.2.1 IETF 292M packet format

In collaboration with the USC Information Sciences Institute (ISI East), a packet format for transporting SMPTE 292M video was proposed to the IETF[vi]. A payload format for

RTP (Real Time Protocol) packets is proposed for SMPTE 292M video, to be carried inside UDP and IP packets. The proposed format is shown in Figure 18 . Extensions to the base RTP packet format are proposed as follows:

1) An extension to the standard 16-bit RTP header sequence number to 32 bits so that the sequence number does not wrap around too quickly.

2) An 11-bit video line number is added to the RTP header to provide easy access to position in the video stream in case packets are lost (this was not in the IETF draft at the time the UNAS system was designed).

3) A 148.5 MHz timestamp is used as the RTP timestamp to allow reconstruction of the timing of the SMPTE 292M stream (the timestamp specification was 10 MHz at the time that the UNAS system was designed).

In addition, it was proposed that related 10-bit luminance and chrominance samples not be separated across packets, and that video data be octet-aligned when packetized. For the UNAS system, an additional criterion was added that the video data be 32-bit word-aligned when packetized in order to make processing of received video data easier.

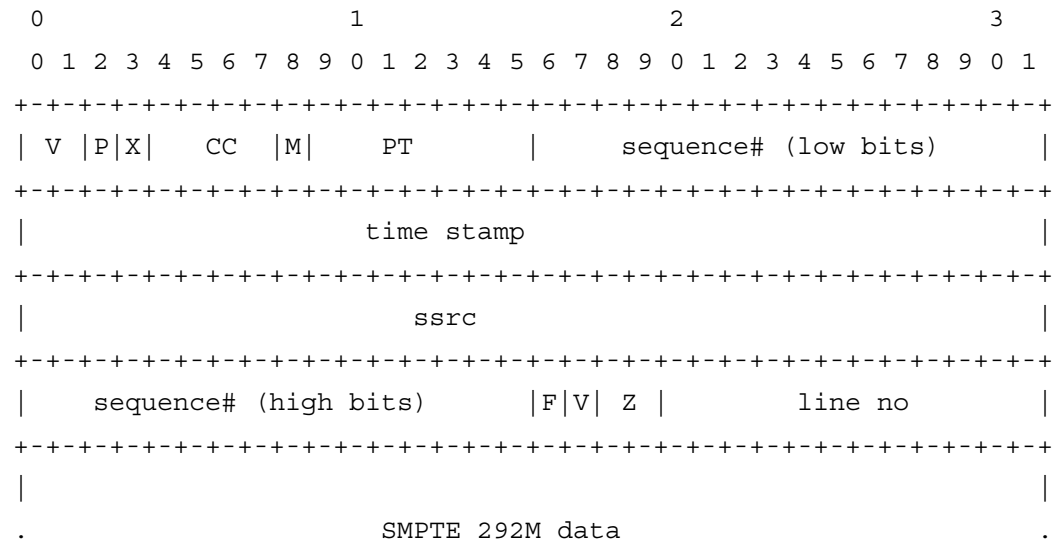


Figure 18 Proposed RTP header format for SMPTE 292M video data

4.6.2.2 292M Video packet assembly

The SMPTE 292M video input code waits for a flag to be set in the VIN_CTL register and the start of a new video frame to start processing video data. Optionally, the frame pulse can be ignored. The data should arrive from the Framer as aligned 20-bit words

from the 292M SDI mezzanine card, so no data alignment is necessary. Some of the 292M HD mezzanine cards have a problem with data alignment due to chip flaws, however, so additional code was added to the FPGA to do bit alignment of the video (enabled via the VIN_CTL register). The following processing steps then occur:

1) A 20-byte IP header is constructed. The user can enter the IP Type of Service (8 bits), IP ID field (16 bits), IP Time-to-Live (8 bits), IP source address (32 bits), IP destination address (32 bits), and the data length of the packet in bytes (16 bits). The data length should be a multiple of 20 bytes, so that an integral number of 20-bit video words are included and the packet ends on a 32-bit boundary. The default data length is 520 bytes, giving an IP data length of 564 bytes. A Start-Of-Packet flag is appended to the first word of the IP header

2) The IP header checksum is calculated in parallel with header construction and appended as the last 16 bits of the IP header.

3) An 8-byte UDP header is constructed. The user can enter the UDP source port (16 bits) and the UDP destination port (16 bits).

4) A 16-byte RTP header is constructed. The user can enter an RTP synchronization source ID (32 bits) and an RTP data type (8 bits). Other header fields are automatically calculated (sequence number, timestamp, video information) and inserted. The format of the header follows that described in reference 1 for SMPTE 292M video.

5) Input video is processed to insure bit alignment, and video type is extracted from the sequencing of timing reference signals.

6) 20-bit video words (10-bit chroma and 10-bit luma) are converted to 32-bit data words in an 8-state machine, and placed in a shallow (64 word) internal FIFO.

7) Data words are read from the internal FIFO and written out into the external hardware FIFO (RFIFO). When a counter decremented from the length field indicates that the end of the packet has been reached, an End-Of-Packet flag is appended to the last data word and header construction is started again.

4.6.2.3 292M Video packet disassembly

The video output side of the Video PLPE withdraws packets from the hardware FIFO (TFIFO) and removes video words to reconstruct the outgoing 292M serial stream. The design waits for TFIFO to reach half full, then begins withdrawing video data at a default rate set by the DDS2 clock (74.250000 MHz for SMPTE 292M). The following operations are then performed on the data stream:

1) IP, UDP, and RTP header information is extracted from the data stream. A checksum is calculated across the IP header information and compared to the last 16 bits of the received header; if they do not match, an error is flagged in the VOUT_FLAGS register and the IP_HDR_ERRS counter is incremented.

2) Packet sequence number checking is done using the 32-bit sequence number contained in the extended RTP header (16 bits in the first header word and 16 bits in the

fourth header word). Counts are kept of sequence number mismatches of ±1, ±2, ±3, and >3, and exported to registers.

3) Data is placed in a shallow internal FIFO (64 words deep). This is necessary so that the video stream is uninterrupted while header processing occurs. Reads are done from the external hardware FIFO (TFIFO) as long as the internal FIFO is less than half full.

4) When the internal FIFO reaches half full, an 8-state machine reads data from the internal FIFO and extracts 20-bit video words (10-bit chroma and 10-bit luma) from the 32-bit wide data.

5) Bit alignment of the output data is checked and the output video type is determined from the TRS signals in the video stream.

6) The 292M video line CRC is calculated and compared to the value found in the video stream after the EAV reference. Errors are reported in the VOUT_FLAGS register and counted in VOUT_CRC_CNT.

4.6.2.4 259M Video packet assembly

The SMPTE 259M video input code waits for flag to be set in the VIN_CTL register to start processing video data. The data arrives from the Framer as aligned 10-bit words from the 259M SDI mezzanine card, so no data alignment is necessary. The following processing steps then occur:

1) A 20-byte IP header is constructed. The user can enter the IP Type of Service (8 bits), IP ID field (16 bits), IP Time-to-Live (8 bits), IP source address (32 bits), IP destination address (32 bits), and the data length of the packet in bytes (16 bits). The data length should be a multiple of 20 bytes, so that an integral number of 10-bit video words are included and the packet ends on a 32-bit boundary. The default data length is 520 bytes, giving an IP data length of 564 bytes. A Start-Of-Packet flag is appended to the first word of the IP header

2) The IP header checksum is calculated in parallel with header construction and appended as the last 16 bits of the IP header.

3) An 8-byte UDP header is constructed. The user can enter the UDP source port (16 bits) and the UDP destination port (16 bits).

4) A 16-byte RTP header is constructed. The user can enter an RTP synchronization source ID (32 bits) and an RTP data type (8 bits). Other header fields are automatically calculated (sequence number, timestamp, video information) and inserted. The format of the header follows that described in reference 1 for SMPTE 292M video.

5) 10-bit video words are converted to 32-bit data words in a 16-state machine, and placed in a shallow (64 word) internal FIFO.

6) Data words are read from the internal FIFO and written out into the external hardware FIFO (RFIFO). When a counter decremented from the length field indicates that the end of the packet has been reached, an End-Of-Packet flag is appended to the last data word

and header construction is started again.

4.6.2.5 259M Video packet disassembly

The video output side of the Video PLPE withdraws packets from the hardware FIFO (TFIFO) and removes video words to reconstruct the outgoing 259M serial stream. The design waits for TFIFO to reach half full, then begins withdrawing video data at a default rate set by the DDS2 clock (27.000000 MHz for SMPTE 259M). The following operations are then performed on the data stream:

1) IP, UDP, and RTP header information is extracted from the data stream. A checksum is calculated across the IP header information and compared to the last 16 bits of the received header; if they do not match, an error is flagged in the VOUT_FLAGS register and the IP_HDR_ERRS counter is incremented.

2) Packet sequence number checking is done using the 32-bit sequence number contained in the extended RTP header (16 bits in the first header word and 16 bits in the fourth header word). Counts are kept of sequence number mismatches of ±1, ±2, ±3, and >3, and exported to registers.

3) Data is placed in a shallow internal FIFO (64 words deep). This is necessary so that the video stream is uninterrupted while header processing occurs. Reads are done from the external hardware FIFO (TFIFO) as long as the internal FIFO is less than half full.

4) When the internal FIFO reaches half full, a 16-state machine reads data from the internal FIFO and extracts 10-bit video words from the 32-bit wide data. The state machine is reset at the start of every packet to maintain synchronization, since packets are constructed in 20-byte multiples and the state machine should repeat the same state every 20 bytes for a continuous video data stream.

5) If the internal FIFO ever goes empty (if the video data stream is interrupted, for example), the video data conversion state machine is stopped and must be re-entered from the start sequence above.

4.6.2.6 Video PCI interface

The Video PLPE PCI interface is similar to the Framer PCI interface. The design is a target-only PCI interface (i.e., cannot act as bus master) written in the Protocol Compiler graphical programming language. The design requires 5 clocks for a PCI transaction (read or write) because additional wait states added to the Framer PCI design were left in the PLPE PCI design to assure reliability. The PCI design uses address bits adr[26:24] to determine the target FPGA; for the PLPE these bits are set to “010”. Addressing for the registers in the PLPE is built into the PCI design block (see the Video PLPE Specification for addresses and descriptions of the registers).

4.6.2.7 Video design verification

Verification of the Video PLPE transmit and receive designs was done using test video data generated in a Protocol Compiler test bench. When the logic was correct, timing

simulations were done using the Quartus FPGA simulator. Finally, testing was done using video test signals in the laboratory. Capture and analysis of standard definition video was done using a Tektronix WFM601 video waveform monitor, and high definition testing was done using a Tektronix WFM1125 HDTV waveform monitor.

4.7 Packet diagnostics

4.7.1 Packet jitter measurements

When carrying high speed real time traffic on an IP packet network, the network parameters that affect performance the most are packet loss and packet delay. Since the video traffic being carried is streaming real time data at rates up to 1.5 Gb/s, data that is delayed for any time greater than the buffer depth on the receive side (approx. 2.5 msec at HD video data rates) will effectively be "dropped packets". In addition, large variation in delay between packets through the network can cause problems with video clock recovery on the receive side if the recovered clock is based on the average data arrival rate as is done here. For these reasons, it was decided to implement packet jitter measurements on live traffic through the network.

Code was written for the Packet-over-SONET receive design that counts clocks between successive packets to determine packet inter-arrival time with a resolution of 13 nsec. Minimum and maximum values of the inter-arrival time are then determined over a period specified by the user and reported in the PLPE registers MIN_SPACE and MAX_SPACE. The feasibility of acquiring and producing histograms of all inter-arrival time data was explored, but was determined to be outside the scope of this project.

4.7.2 Packet sequence number analysis

Dropped packets in a packet flow containing real-time video traffic result in missing video information. Missing information in the recovered video stream will result in glitches in the displayed video; if video timing is lost, the degradation may be as severe as loss of several video frames. Consequently, it was decided to check packet sequence numbers to determine if packets are missing from the stream.

The RTP packet header used for transporting video data contains a 16-bit sequence number in its default header. This was augmented to 32 bits for a SMPTE 292M RTP payload so that sequence numbers do not wrap around too rapidly (a 16-bit sequence number will wrap in approximately 5 seconds for the default packet length of 564 bytes at HD video data rates). The sequence number is extracted from the header and compared to the previous sequence number in the Video PLPE design. Sequence errors are reported in the VOUT_FLAGS register and counted in the VOUT_SEQ_ERRS register.

When packets are dropped by the network, there will be gaps in the sequence numbers. However, sequence numbers will still be monotonically increasing (positive sequence number errors). To test the effect of dropped packets on performance, code was written for the CPE FPGA that drops a packet approximately every five seconds (enabled via the CPE CONFIG register, section 3.19). The loss of a single 564-byte packet was found to

produce visible glitches in the received video. Additional code was written for the Video PLPE that attempted to place "dummy video" in the video stream exactly equal in length to the video that was dropped by a lost packet. The "dummy video" did not contain any video timing reference signals, however, and it was found that this substitution did not improve perceived quality of the received video.

When testing video transport across wide area networks (section 4), it was found that substantial numbers of sequence errors were reported for HD video transport in packets, even though the packet rates at the ingress and egress ports were nearly identical (within 2 packets/sec for a 355,000 packet/sec stream). Additional logic was added to the Video PLPE design to keep separate tallies of packets with sequence numbers both larger and smaller than they should have been. It was discovered that substantial re-ordering of packets occurred at high packet rates. In several experiments across wide area networks it appeared that while no packets were being lost, many packets were placed out-of-order in the data stream (i.e., negative sequence number errors).

4.8 Gigabit Ethernet

4.8.1 Gigabit Ethernet design objectives

Gigabit Ethernet (GbE based on IEEE802.3, IEEE802.3z) and 10 Gigabit Ethernet (10GbE based on IEEE802.3ae) are significant protocols for the present and the future. Modern day organizations depend critically on their local area networks (LANs) to provide connectivity of a growing number of mission critical data and communications services. The ubiquitous 10Mbps 10BASE-T has already been supplanted with the leading choice, 100BASE –T, which has ten times more transport capacity. This growing number of 100BASE-T connections has created the need for even higher speed network technology at the backbone and server level. This is being provided in a smooth upgrade path through the use of Gigabit Ethernet. This trend for yet another order of magnitude improvement is already driving metro and some core networks to roll out 10GbE transport services.

The UNAS System Requirements Document states Gigabit Ethernet, following IEEE802.3, full duplex optical transport over fiber, will be implemented. This requires implementation of the following sublayers of the Physical and Data Link layers according to 803.3:

- PMA - Physical Medium Attachment. PMA provides conversion of data path width to and from ten bits if needed, ten-bit code alignment
- PCS - Physical Coding Sublayer. The PCS function accepts MAC framed packets from the GMII interface, provides 8B/10B encoding and encapsulation into a code-group stream in the transmit direction, as well as decoding and extracting MAC framed packets from the received code group stream. The PCS layer provides the ability to communicate with the link partner equipment through the Auto negotiation functions.

- GMII - Gigabit Media Independent Interface. The GMII is designed to make differences among the various media transparent to the MAC sublayer.
- MAC – Media Access Control. The MAC layer encapsulates user data within a MAC Frame, verifies the correctness of a received MAC Frame, including address filtering and discarding of packets not intended for the receiving MAC, and extracts user data from valid received frames. This device may optionally provide a Pause Control feature.

Because the GbE function shares the network interface mezzanine card with SONET receive and transmit functions, this design is required to handle all necessary clocking and signal path width conversions.

4.8.2 Summary of Work

4.8.2.1 Design Approach

The Gigabit Ethernet feature for the NGI UNAE was designed using the Verilog hardware descriptive language. Verilog was chosen because of its portability, because it lends itself to a structured, hierarchical design methodology, for it's ease of simulation, because of control it provides over synthesis results, and because of the availability and quality of Software tools to support this language.

The design structure itself is highly segmented and hierarchical in nature, in order to facilitate the re-use of repeated functional blocks, as well as to facilitate maintenance and modification. At the top level, the functions unique to Gigabit Ethernet are implemented in three FPGA's. These FPGA's are the MAC, the GMII, and the PMA-PCS FPGA. Each FPGA is further segmented into task specific blocks. For details on these functional blocks, see sections below.

The design flow using the Verilog language consists of code and test bench generation, RTL simulation, front-end synthesis, back end synthesis, and timing-based simulation. After determining specific requirements for the design, the functions necessary to meet these requirements was divided logically, in a way that facilitates simulation, synthesis, and a logical structure that groups functions in a sensible manner. Once this is done, the synthesizeable code is written. For this design, the approach was to take each small function, write the synthesizeable code describing one or a group of modules, which provide a specific function. Each of these specific functions was then verified by writing a test bench which, through RTL simulation, verifies the intended behavior. The portion of the design was then synthesized in a stand-alone manner, in order to address any timing or synthesizeability issues at this point. Each section of the design was performed in this manner, typically beginning at the input of the transmit data path, and working through to the output. As each portion of the design is completed it is verified with all prior portions. Through this approach, each portion of the design is verified specifically while leveraging both design and simulation test bench efforts towards a top-level application.

4.8.2.2 Simulation and Test Benches

The Gigabit Ethernet design was simulated using a test bench created using the Verilog language. There were four stages of simulation and verification performed during the design and debug of this system. First individual portions of each FPGA in the system (see section 3.15 for details on the Gigabit Ethernet system) were simulated with a functional (RTL) test bench designed to verify that specific portion of the design. Second, an RTL simulation was performed on each FPGA in the system. Third, the entire system was functionally simulated using a top level test bench. Finally the entire Gigabit Ethernet system design, consisting of three FPGA's, was simulated using a Verilog model for each FPGA which included back-annotated timing information.

This final test bench was designed to simulate as closely as possible the laboratory operating conditions, and included a behavioral model of the hardware FIFO utilized on the UNAE main board. The final test bench was structured such that it consisted of a top level test bench, a stimulus generation module, and various tasks which, when invoked by the stimulus module, forced stimulation of the design in a desired manner, and verified the resulting behavior of the design. The capabilities of this test bench include simulation of PCI bus cycles, as well as the creation of user traffic which was placed into the FIFO in the Transmit direction (coming from the backplane). The capability to execute bus cycles was utilized in a software-like manner to configure the system. The traffic generation capability could then generate traffic in the transmit direction. The transmit output of the design was tied to the receive input of the design, in the manner that a fiber loopback would use transmit data as receive data, and the system thus tested. The test bench had the capability to vary the packet rate, the inter-packet gap, and the contents of the packets. It also possessed the capability to extract received packets from the FIFO model, and verify the contents. Through the PCI interface, the test bench was also able to verify statistics gathering capability. The structuring of the test bench with various tasks to perform various functions allowed the testing to be quickly and easily modified, testing any or all functions of the design in any given simulation run. It also provided for a self checking test bench, such that it was not necessary to look at waveforms to interpret test results. The results could be dumped to the screen for immediate review, or stored to a file for later retrieval.

MAC

Functional Overview

The Media Access Control (MAC) framing functions for the Gigabit Ethernet Interface implementation on the UNAE hardware are provided by the FPGA designated the PLPE for both POS and Video applications, referred to in this document as the MAC Framer. The hardware functions within the MAC Framer are limited to the MAC layer functions as defined in section 4 of IEEE 802.3, a FIFO interface, and a modified version of the Reconciliation Sub-layer Gigabit Media Independent Interface (GMII) as defined in section 35 of the same document. Additionally, a simple PCI interface allowing microprocessor access to a set of internal registers used for configuration and monitoring

is provided. The PCS layer functions (i.e. 8B/10B, auto-negotiation) are provided on the mezzanine by the PMA-PCS FPGA, and so are not defined in this section.

Figure 19 shows a functional block diagram representing the functions provided by the MAC Framer.

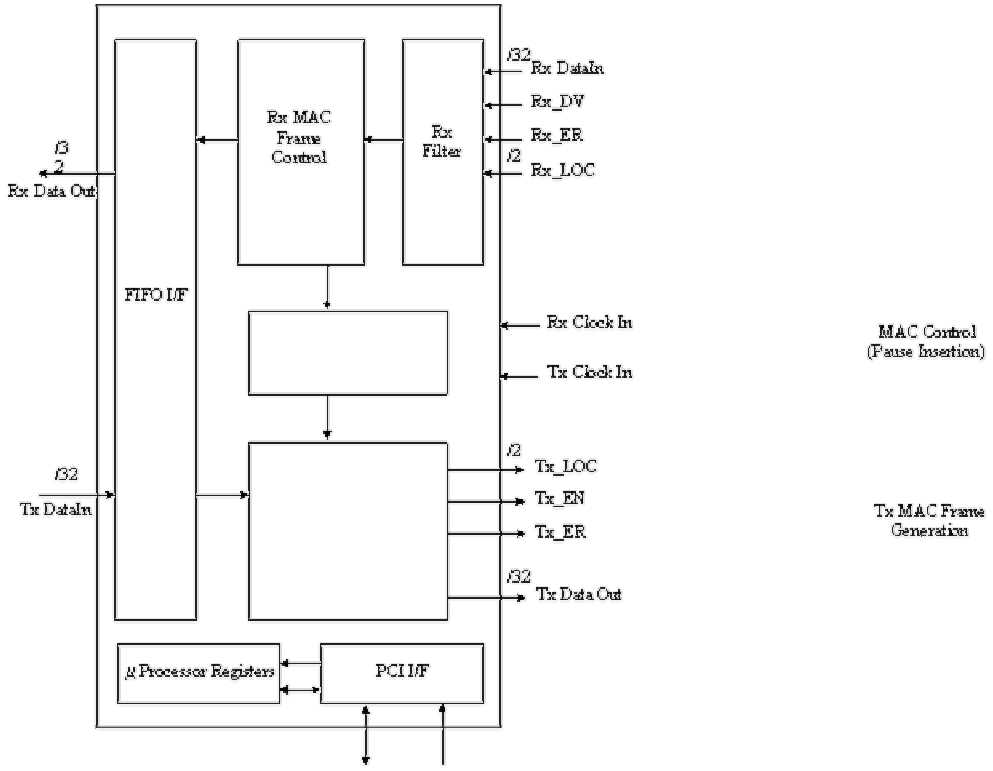


Figure 19 Functional blocks of MAC framer

The MAC layer functions provided consist of the encapsulation of user data within a MAC Frame, the verification of the correctness of a received MAC Frame, including the identification and discarding of packets not intended for the receiving MAC, and the removal of user data from correctly received frames for presentation to the FIFO interface. This device may optionally provide a MAC Control (Pause Control) feature that is not implemented. Each of these operations is described in detail in the following sections.

Transmit FIFO interface

The transmit FIFO interface performs two primary functions. First, under control of the microprocessor register set, the FIFO is configured to operate in the desired mode. Second, the interface performs FIFO read cycles when necessary, and monitors the TAG value to determine the beginning and end of user data packTets. These packets are then

handed off to the transmit framer state machine.

Upon the detection of the FIFO asserting the half full flag, the Read FIFO state machine begins performing read cycles to extract data from the FIFO. In normal operation, data is continuously read until an end of packet TAG value is detected. In this manner, data is read from the FIFO one entire packet at a time. Since the CPE will not write data into the FIFO that is not a part of a user packet, the first word read from the FIFO after assertion of the half full flag should always be the first word of a packet, as indicated by the SOP value contained within the corresponding TAG bits (see Table 2 in section 4.3.2.2). The state machine verifies this value for each read packet operation. If this value is not correct, an error is assumed, and data is read from the FIFO until an EOP TAG value is encountered, thus re-establishing the Packet boundary. In this case of an errored SOP TAG value, the contents of the Transmit Errored Packet Count register are incremented, and data is read from the FIFO and discarded up until and including the word concurrent with the next EOP TAG.

Transmit Framer State Machine

The transmit Framer state machine accepts user packets from the Transmit FIFO interface, and encapsulates them within a MAC FRAME, as shown in Figure 20 and Figure 21. The frame is constructed as it is transmitted, and is transmitted most significant byte first, as it is read from the FIFO. The frame construction operation utilizes a set of register stages, which it pre-loads with the portions that are pre-determined of the MAC frame to be transmitted before the user data. The fields pre-loaded in this manner include the Preamble, the Start Frame Delimiter (SFD), the Destination Address, and the Source Address. This pre-loading occurs at the point after power up and configuration when the transmit enable bit of Master Control Register is set, as well as immediately after the transmission of the last word of a packet.

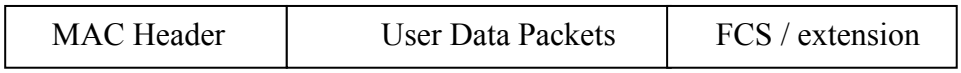


Figure 20 Gigabit Ethernet user data encapsulation

The first part of the MAC frame is the preamble. A field containing 0x55 in each of its seven octets, this portion of the frame is placed into the first two stages of a set of frame construction registers. Next, the Start Frame Delimiter (SFD), which contains a value of D5h, is placed in the most significant byte position of the second register stage. The third, fourth, and fifth stages are then preloaded with the contents of the Destination and Source Address registers. At this point, the pre-loading function is complete, and the Transmit Framer State Machine will remain idle until the FIFO indicates a packet is ready for transmission.

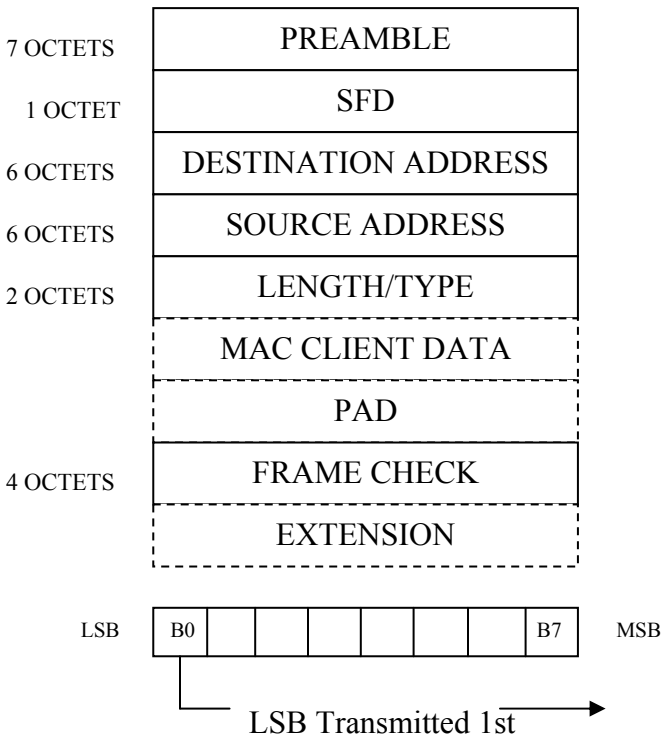


Figure 21 Gigabit Ethernet MAC frame structure

When the Transmit Framer State Machine detects that the FIFO half full flag has been asserted, it resumes the process of constructing the MAC Frame. The first word of user data is read from the FIFO, and the contents of the TAG are verified to be the value associated with a SOP (see Table 2, FIFO packet tags indicating packet data type). If the TAG value is incorrect, the FIFO is read, effectively dumping the data, until an EOP TAG is encountered. At this time the State Machine returns to the idle state until the half full flag is once again asserted. If the TAG value is correct, the first data word from the FIFO is placed into the 6^{th} and 7^{th} register stages, such that the least significant two bytes are placed into the most significant two bytes of stage 6, and the most significant two bytes are placed into the least significant two bytes of stage 7. The least significant two bytes of the first data word read from the FIFO represent the length field (in the case of IP packets). This value is also copied into the least significant two bytes of stage 6, and constitutes the Length Field of the MAC frame. It should be noted that this implementation does not currently support non-IP packet types, nor does it support packets of length greater than 1518 bytes. The second word of the IP packet is then read from the FIFO and placed in the 7^{th} and 8^{th} register stages, as are all subsequent packet words. As each user data word after the 2^{nd} is read from the FIFO and placed into the 7^{th} and 8^{th} stages, the data currently stored in each staged is loaded into the corresponding lower numbered stage, with the first staged shifted into the Transmit GMII interface. In the case that any IPG limitations resulting from this architecture are deemed

unacceptable, two frame construction register sets will be used in an alternating fashion. In this manner, the second register set will be pre-loaded while the first is transmitting a packet.

Data is continually read out of the FIFO and transferred to the Transmit GMII, until the EOP TAG value is encountered. When this occurs, reading from the FIFO stops. If the packet length is less than the MinFrameSize, and PAD insertion is enabled, a number of PAD data octets sufficient to increase the frame length to MinFrameSize is loaded into stages 7 and 8. Although not currently supported, the PAD data insertion function will allow insertion of PAD octets only in increments of four. Once the packet data and any required PAD data is loaded, the calculated FCS value is loaded into stages 7 and 8. Finally any necessary extension octets are inserted, and the remaining packet words contained in the register stages are shifted to the Transmit GMII interface for transmission. Once the entire packet has been shifted out of the register stages, the state machine again pre-loads the stages with the pre-determined fields, and stands idle until the half full flag is asserted, beginning the process again.

The FCS value is calculated over the contents of the source address, the destination address, length, and LLC data and PAD octets. The FCS is a four-octet (32bit) value calculated using the following polynomial.

$G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1$

The resulting FCS word is placed into the registers such that it is transmitted in the order x31, x30, ….x1, x0.

4.8.2.3 Gigabit Media Independent Interface (GMII)

The MAC FPGA implements a modified version of the GMII as defined in section 35 of IEEE 802.3. The interface is modified for this application in two ways. The first change is the use of a 32-bit interface instead of an 8-bit interface. This requires the use of the tx_loc[1:0] and rx_loc[1:0] signals to indicate the location of the change indicated by changes in the status of the tx_en and tx_er signals. This also imposes some limitations on lengths of extensions and errors, as transitions in the combined state of these control signals can only occur once in every four octets of data. Second, the clock configurations are modified in order to support the 32-bit interface (31.125 MHz instead of 125). Clocks are driven from the PMA-PCS FPGA to the MAC FPGA**Error! Bookmark not defined.**, for both transmit and receive operations, in order to work within the context of the clock distribution architecture of the UNAE hardware.

Transmit GMII

The transmit GMII interface accepts MAC frames from the Transmit Framer State Machine, and creates the tx_en, tx_er, and tx_loc signals. The tx_loc signal is used to tell the PMA-PCS FPGA logic which octet within the 32-bit data word corresponds to the change in signal status indicated by the tx_en and tx_er signals. The possible encodings of the tx_en, tx_er, tx_data<7:0> are shown in Table 3, with each byte of the 32-bit interface representing a separate value of tx_data<7:0>.

Table 3 Transmit GMII control signal indications

TX_EN	TX_ER	TXD<7:0>	Description
0	0	00 - FF	Normal Inter-frame
0	1	00 – 0E	Reserved
0	1	0F	Carrier Extend
0	1	10 – 1E	Reserved
0	1	1F	Carrier Extend Error
0	1	20 – FF	Reserved
1	0	00 – FF	Normal data transmission
1	1	00 - FF	Transmit Error propagation

Table 4 Transmit GMII control signal indications

TX_LOC[1:0]	TX GMII Data Status
00	All four bytes are valid data
01	Only the most significant byte is valid data.
10	Only the most significant 2 bytes are valid data.
11	Only the most significant 3 bytes are valid data.

Carrier extensions are not supported in this implementation, as they are defined only for half-duplex operation. Transmit error propagation is indicated by the Mac to the PCS layer via the GMII to request that the PHY deliberately corrupt the contents of the frame such that the receiver will detect the corruption and discard the frame. This is requested by the MAC Transmit GMII in the case that an error occurs during the construction of the MAC frame. Figures 35-3 and 35-4 in IEEE 802.3 depict the behavior of the transmit GMII signals.

Receive GMII Interface

The Receive GMII accepts MAC Frames from the PMA-PCS FPGA, via the GMII Bridge FPGA, using the rx_dv, rx_er, rx_data, and rx_loc, to delineate the frame within the received data stream. The Receive GMII implemented is modified from the implementation defined in section 35 of 802.3 in the same manner as the Transmit GMII.

The operation of the Receive GMII is limited to normal inter-frame indication, false carrier indication, normal data reception, and data reception error. Permissible encodings as defined in IEEE 802.3, are shown in Table 5. Encodings other than those indicating in

the list of supported operations above, however, are not supported and shall be treated as errors. Normal frame reception is depicted in figure 35-8 of IEEE 802.3. Reception with error is depicted in Figure 35-11 of IEEE 802.3. False carrier indication is depicted in Figure 35-12 of IEEE 802.3.

Table 5 Received GMII Operation

Rx_dv	Rx_er	Rx_data<7:0>	Description
0	0	00 – FFh	Normal inter-frame
0	1	00h	Normal inter-frame
0	1	01 - 0Dh	Reserved
0	1	0Eh	False Carrier indication
0	1	0Fh	Carrier extend
0	1	10 – 1Eh	Reserved
0	1	1Fh	Carrier Extend Error
0	1	20 – FFh	Reserved
1	0	00-FFh	Normal Data Reception
1	1	00-FFh	Data Reception Error

Receive Media Access Management

The Receive Media Access Management detects the beginning of a received frame present on the Receive GMII when the rx_dv signal is asserted (see Figure 35-8 of IEEE 802.3). If no Data reception error is indicated by a simultaneous assertion of the rx_er signal, the packet is passed on to the address filtering logic. If address filtering is enabled (see Master Control Register) the DA field of the received frame is verified against the programmed value for the MAC-FPGA's local MAC address (contents of the SA Registers). If it is determined that the frame is intended for this MAC, the user data is aligned to a 32-bit boundary according to the contents of the rx_loc signal states during the first GMII bus cycle in which the rx_dv was asserted, and then passed on to the Receive Framer State Machine. If the DA field does not match, the incoming frame is discarded. If address filtering is disabled, the received frame passes through the filter logic to the Receive FCS Verification logic, regardless of the DA field contents.

Receive FCS Verification

For each frame received, a new FCS value is calculated. When this new FCS calculation is complete, the resulting value is compared to the value contained within the received frame's FCS field. If this comparison verifies that the received FCS value is correct, no action is taken. Should the received FCS valued be determined to be incorrect, the error

is reflected by incrementing the contents of the Receive FCS Error Count Register. The FCS Error Count register will also be incremented in the case that both the RX_DV and RX_ER signals are both assisted during frame reception.

Receive Framer State Machine

Regardless of the outcome of the FCS verification procedure, the receive framer state machine strips all header information, the FCS, any PAD data, and any extension data off of the incoming frame and hands the MAC Client Data portion of the frame to the receive FIFO interface for storage in the external First-in-first-out (FIFO) memory on the UNAE.

Receive FIFO interface

The receive FIFO interface accepts the MAC Client Data portion (IP packet) of the received frame from the Receive Framer State Machine, and loads it into the FIFO. The Receive FIFO interface also generates the TAG data field values associated with the status of each word of the packet data written into the FIFO. See Table 2 for details on specific values of the TAG data. The received packet and TAG data are written into the FIFO simultaneously by asserting the rx_fifo_wen signal. This signal remains asserted until the entire packet is written into memory.

FIFO Configuration

The receive FIFO configuration programming is under control of the MAC FPGA. Although the current implementation provides no mechanism by which to program the preset levels of the partial-full flags, the required signals are available within the FPGA. These configuration levels are currently used in their default mode. The Master Reset, Partial Reset, and FWFT signals are under microprocessor control via the Reset and FIFO Configuration/Status registers. The value corresponding to the desired mode of operation should be written to the FWFT bit in the FIFO Configuration/Status before a Master Reset for the FIFO is de-asserted in order to configure the FIFO. The bit defaults to one, selecting the FWFT operation mode if no write operation is performed. Transmit FIFO configuration is under control of the CPE FPGA. The CPE FPGA specification should be consulted for the specifics of Transmit FIFO configuration.

MAC Control

The MAC Control/Pause insertion functions are not supported in the initial version of this implementation, but may be added as an extension.

4.8.2.4 Microprocessor Interface

The microprocessor interface is provided as a target only, non-burst, PCI interface accessing a set of control and statistics registers. A subset of the functions defined in the PCI local Bus Specification are supported, including Memory Read, and Memory Write cycles. Interrupt Acknowledge, Special Cycles, I/O read and write cycles, configuration read and write cycles, burst cycles, and Memory write and invalidate cycles are not

supported in this implementation. For specific information on the access cycles supported, refer to the PCI Local Bus Specification, version 2.2.

4.8.2.5 Design Hierarchy

The PMA-PCS FPGA was designed in Verilog, and consists of twenty-one functional blocks. The structure created by these functional blocks is depicted in . The specific function of each modular block is described below.

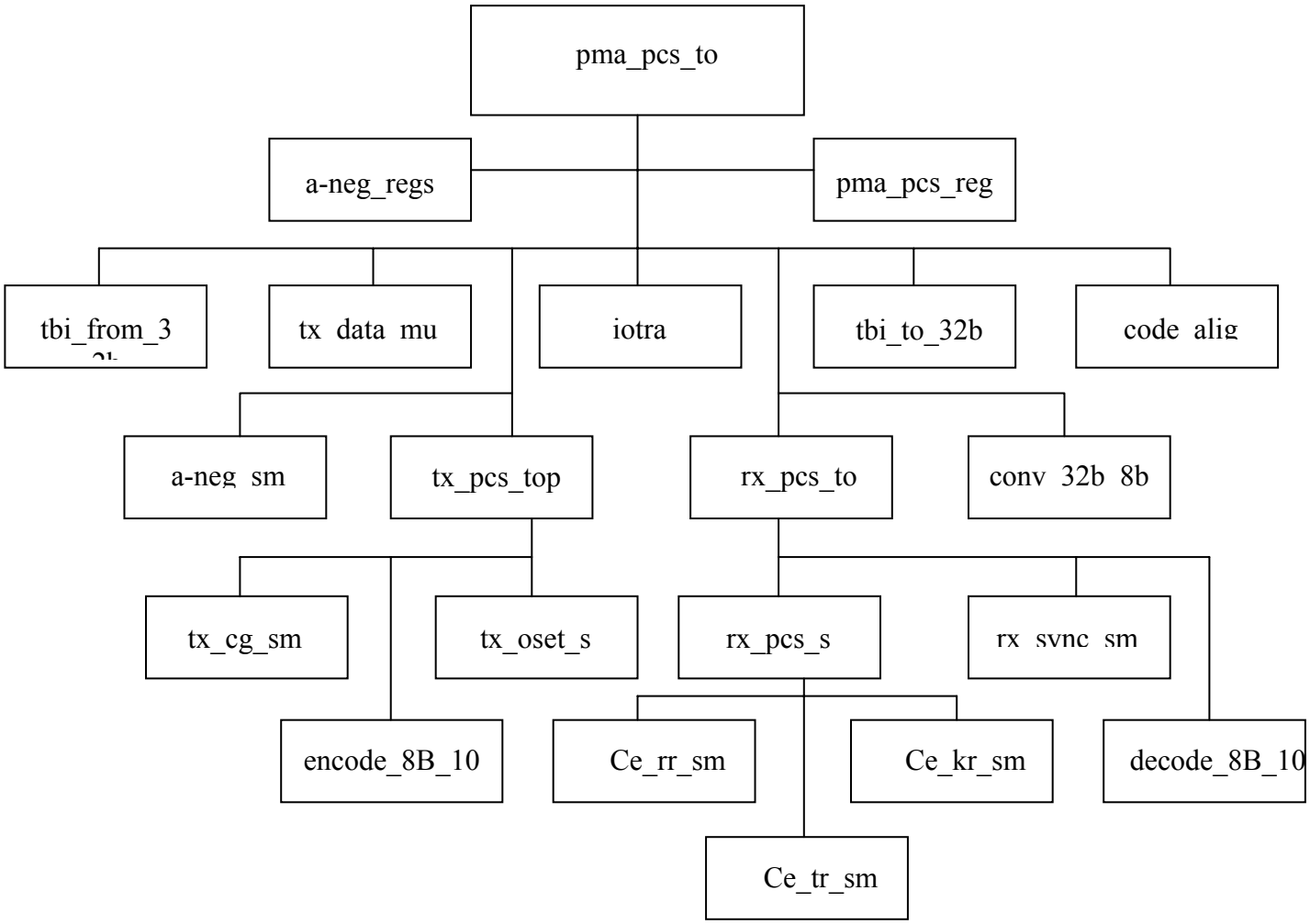


Figure 22 Gigabit Ethernet PMA-PCS FPGA design hierarchy

pma_pcs_top:

This is the top level module, instantiating and providing interconnect for mid-level and lower-level modules. This module also instantiates the DCM's (Digital Clock Managers) used to generate internal clocks, and distributes these clocks and their associated reset

signals. A description of each of these clock domains is provided in the section on Clock Distribution. This module defines I/O buffers and, and registers data as it enters and leaves the part. Finally, this module controls data flow and clock output selection based upon the setting of the mode select bit as defined in the PCS Configuration Register.

aneg_regs:

This module provides the registers associated with the Auto Negotiation portion of the PCS functions provided for Gigabit Ethernet. The structure and usage of these registers is defined in clause 37 of IEEE 802.3

pcs_regs:

This module provides the register set used to control and monitor all functions of the part

Iotran:

This module provides an interface between data in a 32-bit format and the SONET transceiver. In a SONET mode of operation, this 32-bit data is exchanged directly with the mezzanine connector data bus. In a Gigabit Ethernet mode of operation, 32-bit data is exchanged with the input/output of the path width conversion FIFO's utilized in the tbi_from_32b and tbi_to_32b modules. In addition, due to the difference in the order of bit transmission between the two supported protocols, the conversion must be adapted to the selected protocol.

tx_pcs_top:

This module instantiates the three lower level modules (tx_cg_sm, tx_or_set_sm, and encode_8B_10B) that together form the Transmit PCS functions.

rx_pcs_top:

This module instantiates the lower level modules that together perform the Receive PCS functions necessary for GbE operation.

tbi_from_32b:

This module, instantiated by the pma_pcs_top module, provides the FIFO and associated control logic necessary to convert received 32 bit data from the iotran module to a ten-bit wide format to be provided to

the code_align module.

tbi_to_32b:

Instantiated by pma_pcs_top, this module provides the FIFO and associated logic necessary for conversion from the ten-bit wide data path output of the 8B/10B encoder to a 32-bit format for presentation to the iotran module.

ce_kr_sm, ce_rr_sm, ce_tr_sm:

These modules together form the check_end function, which is a portion of the PCS Receive state machine. Each module of check_end provides an indication of the possible end sequences indicated by the name – i.e. ce_kr_sm checks for end sequences which have |K|R| as the first two code groups of the sequence.

code_align:

This module provides the ten-bit alignment of the received code words based on the detection of a Comma within the incoming bit stream.

conv_32b_8b:

This module provides a conversion of data from the 32-bit format, provided across the mezzanine connector interface, into an 8-bit format provided to the PCS functions, such as 8B/10B encoding.

rx_pcs_sm:

This module implements the PCS receive state machine as depicted in Figure 36-7a/b in IEEE 802.3.

rx_sync_sm:

This module implements the Receive Synchronization state machine as depicted in figure 36-9 of IEEE 802.3.

tx_cg_sm:

This module implements the transmit code-group state machine as depicted in figure 36-6 of IEEE 802.3.

tx_data_mux:

This module provides data path selection based upon the selected mode of operation. In addition, this module creates the correct bit transmission sequence. This is necessary due to the LSB first transmission of GbE data, as opposed to the MSB first nature of transmission of SONET data.

tx_or_set_sm:

This module implements the Transmit Ordered Set State Machine as depicted in Figure 36-5 of IEEE 802.3.

4.8.2.6 Gigabit Ethernet mode PMA-PCS FPGA

Functional Overview

The PMA-PCS FPGA provides the Physical Coding Sub-layer (PCS) and Physical Medium Attachment (PMA) functions for the Gigabit Ethernet operation of the UNAE hardware when configured for Gigabit Ethernet mode. It must also provide 16 to 32 bit data path width conversion, as well as clock division, when configured to support the SONET mode of operation.

The PMA functions are required because the IC's on the mezzanine do not perform these operations, as a standard Gigabit Ethernet PHY would. These include the conversion of data path width to and from ten bits (SONET transceiver provides a 16-bit interface), as well as the ten-bit code alignment when in Gigabit Ethernet mode. In SONET mode, the part performs a 16 to 32-bit conversion. It then bypasses all PCS functions, which are specific to GbE, and provides/accepts 32-bit data to/from the mezzanine Connector.

The PCS function provided for Gigabit Ethernet operation accepts MAC framed packets from the GMII interface, provides 8B/10B encoding and encapsulation into a code-group stream in the transmit direction, as well as decoding and extraction of MAC framed packets from the received code group stream. The PCS layer also provides the ability to communicate with the link partner equipment through the Auto negotiation functions.

Transmit Data Path

When configured for a Gigabit Ethernet operation, transmit data is registered into the PMA-PCS FPGA in a 32-bit, 31.25MHz bus from the mezzanine connector. This data path essentially functions as a modified GMII, in that data is presented on a frame-by-frame basis, along with a TX_EN and TX_ER signal. In addition, a tx_ctl_loc signal is provided across this interface in order to indicate the location within the four-byte field of the event indicated by any change of the combined state of the TX_EN and TX_ER signals. Once registered within the part, the combination of data and control signals is passed on to the conv_32b_8b module. Here the data and control signals are converted to an 8-bit, 125 MHz format to be provided to the Transmit PCS logic in a standard 8-bit GMII format. The permissible encodings of TX_EN, TX_ER, and TX_DATA<7:0> are recreated, from Table 35-1 in IEEE 802.3. , in Table 4 Transmit GMII control signal indications.

The Transmit Ordered Set State Machine interprets the data and control inputs from the conv_32b_8b module, and generates ordered sets as defined in Table 36-3 of IEEE 802.3. The generated ordered sets are then encoded into 10-bit code groups by the combined operation of the Transmit Code-Group State Machine and the encode_8B_10B module. The result of the operation of these Transmit PCS functions is the encapsulation of MAC-framed packets within a contiguous stream of 8B/10B encoded code-groups, with idle pattern ordered sets being transmitted in the absence of a packet. The Transmit Ordered Set State Machine and Code-Group state machines also support the transmission of configuration data. The Auto Negotiation function provides a XMIT control signal to

these state machines, which can take on one of three values. These values are: configuration, idle, and data. When XMIT=idle, the transmit PCS function provides a continuous stream of |I| (idle) ordered sets to the PMA device for transmission. When XMIT=configuration, the transmitted data stream consists of an alternating pattern of the two, four byte (|C1| and |C2|), configuration ordered sets. In the case that XMIT = DATA, the Transmit PCS places encapsulated MAC frames within an otherwise continuous transmission of idle ordered sets, as previously described.

The continuous encoded data stream provided by the Transmit PCS function takes the form of a ten-bit bus parallel data interface operating at 125MHz. In most Gigabit Ethernet implementations, this ten-bit interface would constitute the connection between the MAC chip, which typically provides the PCS functions, and the PHY chip, which provides the PMA and PMD operations. In this application, however, the requirement is for the hardware to support both Gigabit Ethernet and SONET modes of operation. As a result of this requirement, the PHY chip chosen provides the SERDES and CDR functions via a 16-bit interface. In order to operate correctly, the 10-bit data interface must be converted to a 16-bit format, then provided to the PHY chip at a 1250Mbit/s line rate, such that the LSB is transmitted first. Because the clock frequency required to create 1250Mbit/s in a 16-bit format (78.125MHz) is a non-integer multiple of the frequency required to create 1250Mbit/s in a 10 bit format (125MHz), clock frequency synthesis must be used to generate the proper clocks. This clock synthesis process (see section 4.8.2.10 Clock Distribution.) results in a situation where no known edge relationship exists between the resulting clocks, even though one is directly derived from another. In order to overcome issues created by the resulting clock , and to provide for data path width conversion, an extremely wide FIFO is employed. The FIFO width was chosen to be 160 bits, because 160 is the lowest common denominator between 10 an 16, thus avoiding the need to perform partial word writes. The transmit FIFO control logic performs a write operation once every 16 125MHz clock cycles. Once a certain fill level has been reached, the logic performs a read operation once every 5 39.0625MHz clock cycles. Data is handled in a 32-bit format in order to be able to utilize the same circuitry in the iotran module used in a SONET mode of operation.

The 32-bit data enters the iotran module, and is registered, before being converted to a 16-bit 78.125MHz format. Transmit output data is driven via the LVPECL differential drivers provided by the FPGA to connect with the SONET transceiver.

Receive Data Path

16-bit data is received from the SONET transceiver at 78.125MHz via the differential LVPECL inputs on the FPGA. Once received, data is registered, and then converted to a 32-bit 39.0625MHz format. Once in a 32-bit format, received data is clocked into the Receive FIFO by the associated logic 160 bits at a time, once every 5clock cycles. After exiting a reset state, the FIFO is allowed to fill to a specified level. Once this level has been reached, data is read from the FIFO by the controlling logic once every 16 125MHz clock cycles, thus providing a received data path ten bits wide operating at 125MHz.

Because of the previously mentioned design requirements, the received data has no alignment with respect to the ten-bit word. The PMA-PCS FPGA must perform this alignment function, which would normally be provided by a GbE PHY chip. The code alignment module implements this function. Essentially, this is accomplished by two parallel sets of 10-bit wide and ten word deep register stages. One of the register stages serves as a search mechanism. Each possible alignment of the incoming data is created by shifting the alignment a single bit at each stage. The Comma pattern, which is defined to occur only in the |K28.5| code group specifically for this purpose, is searched for in each stage. When this pattern is found in its proper alignment, a comma detect event occurs. If the logic is enabled to detect commas (this depends upon both the user configuration and the state of the Receive Synchronization State Machine), the detected new alignment value is registered. This registered alignment value is a ten-bit representation, where each bit indicates the need of the corresponding stage of the second set of registers to shift the bit alignment, or not. Thus, if shifting the incoming data 3 times creates the proper bit alignment, the first three bits of this alignment value would be high, and the last seven would be low. The second set of registers, controlled by this value would shift the received data the proper number of bits, and then pass it unchanged through the remaining stages.

Once the received ten-bit code groups have been properly aligned, they are decoded into 8 bit words by the 10B/8B decoder. The Receive PCS State Machine and the Receive Synchronization State Machine then interpret the byte-wide data. The Receive Synchronization State Machine establishes the code-group synchronization by monitoring the occurrence of commas within the received bytes, and verifying their proper sequencing. The Receive PCS state machine extracts MAC framed packets from the continuous received byte stream. The received packets are then transmitted, along with the RX_DV and RX_ER signals generated, to the data path conversion logic. Provided by the conv_32b_8b module, this logic performs a 8 to 32-bit path width conversion, converts the RX_DV and RX_ER from 125MHz to 31MHz signals, and creates the RX_LOC signal that indicates which of the 4 bytes within a 32–bit word corresponds to the SOP, EOP, or error indicated by the status of the RX_DV and RX_ER signals. Once in a 32-bit format, the received data is selected by the mode selection mux, registered at the output block, then driven off chip as the received data.

4.8.2.7 Auto Negotiation:

Functional Overview

Auto negotiation for Gigabit Ethernet provides a means for a local device to advertise it's operational modes to a link partner and for the link partner to advertise it's operational modes to the local device. The two devices communicate, via handshake messages, to establish the common mode of operation to be used on the link. The PCS layer provides this ability to communicate with the link partner for Auto negotiation.

Ordered Sets

Auto Negotiation function exchanges information between two devices that share a link segment and automatically configures the two devices according to their abilities. Auto Negotiation is performed using /C/ and /I/ ordered sets (defined in IEEE 802.3 Clause 36). Upper layer protocols or packet data are not allowed to communicate on the link until Auto Negotiation has completed.

Start and Restart

The Auto Negotiation function must be enabled under the following conditions:

- The link is initially connected.
- Either device is powered up.
- Either device is reset.
- Either device has been issued a renegotiation request.

During operation, receipt of an Auto Negotiation packet will restart the negotiation process to re-establish the link. At this time, the data path will be disabled by the Auto Negotiation software.

The Auto negotiation implementation for NGI Gigabit Ethernet supports only the basic features necessary to interoperate with similar equipment. This includes the exchange of base page messages indicating equipment capabilities, but not Next page operation. Pause control features are also not supported.

The implementation of the auto-negotiation was divided between hardware and software in order to provide a more flexible solution, as well as to best make use of resources available. The role of the hardware portion of the feature is to insert base page messages into Configuration (|C|)ordered sets in the transmit direction and to extract them from the incoming bit stream in the receive direction. When a new message is received three consecutive times without error, the hardware registers the received message, and asserts an interrupt to software indicating that a new message is available. The role of the software in the system is to process the received messages and generate the transmit messages via the implementation of the auto-negotiation state machine as defined in IEEE 802.3 Figure 37-6.

4.8.2.8 SONET Mode PMA-PCS FPGA

Transmit Data Path

When the part is configured to support SONET operation, the transmit data path flows directly from the input registers to the data selection mux which selects the data to be input to the iotran module. The iotran module performs a path width conversion from 32-bits to 16, then drives the transmit data out using the LVPECL output buffers at 155.52MHz.

Receive Data Path

Received SONET data is brought into the part at 155.52MHz via LVPECL input buffers. It is then converted to a 32-bit path width. The received SONET data is provided to a mux controlled by the mode select bit, then registered and driven out towards the mezzanine connector on rx_data_out[31:0].

4.8.2.9 Microprocessor Interface

Access is provided to internal registers in the PMA-PCS FPGA via an 8-bit microprocessor interface. The interface consists of, eight address lines, eight data lines, a chip select line, and separate read and write enable lines.

4.8.2.10 Clock Distribution.

Clock distribution within the PMA-PCS FPGA is greatly complicated by the need to support both SONET and GbE modes of operation with a single PHY. Conversion from 16 to 32-bit bus width in a SONET mode of operation requires the generation of half rate clocks for distribution on to the UNAE board (both transmit and receive clocks are driven from the SERDES to the PMA-PCS FPGA, and then from the FPGA to the UNAE). In addition, because the PHY chip provides a 16-bit data bus, which must be converted to a ten-bit format for GbE PCS functions, the M/N Clock Frequency Synthesis functions provided in the Xilinx DCM is used to generate the necessary non-integer multiple clock (125MHz generated from 78.125MHz). This 125MHz clock is then divided by four to provide data in a 32-bit wide format to the UNAE. The final output clocks are selected, using a mux, from the GbE divide-by-four and the SONET divide-by-two clocks.

By performing all of these functions within the PMA-PCS FPGA, the existing clock circuitry on the UNAE can distribute two clocks (one for transmit and one for receive) to all other FPGA's as designed. This approach also allows this clock to run at 31.125 MHz, which makes the MAC and Framer FPGA design for this application simpler, with much looser timing constraints. The result is a more complicated clock network within the PMA-PCS FPGA, as shown in the PMA/PCS Specification.

The following is a list including a brief description of each of the clocks within the PMA-PCS FPGA:

- **TX_CLK_IN:** This can be either 78.125MHz (Gigabit Ethernet) or 155.52 MHz (SONET) depending upon the mode of operation. This in an input sourced from the AMCC 3057. It provides the input for the transmit 8/5 clock multiplier, as well as the clock for transmit data as it is driven to the 3057.
- **TX_CLK_DIV2:** This clock is derived from the TX_CLK_OUT, and will be either 39.0625MHz (Gigabit Ethernet) or 77.76MHz (SONET) depending upon the mode of operation. In SONET mode, this clock will be selected as the clock for transmit data as it is driven into the part from the mezzanine connector, as well as being driven to the UNAE main card for use as the transmit timing source. In

Gigabit Ethernet mode, this clock is used to extract data from the transmit FIFO, and into the 32 to 16 bit converter.

- **TX_CLK_125:** This clock is used exclusively in Gigabit Ethernet mode. It is derived from the TX_CLK_OUT, using the clock synthesis feature of the DCM in the Virtex2, and is an 8/5 multiple of the 78.125 MHz clock. This clock is provided to the 32 bit to 8 bit converter, the Ordered Set State Machine, the Transmit Code Group State Machine, and the input control logic of the transmit FIFO.
- **TX_CLK_31**: Used exclusively in Gigabit Ethernet mode, this clock is a divide by 4 derivative of the TX_CLK_125. When in Gigabit Ethernet mode, it provides the input clock for the 32 bit to 8 bit conversion logic, as well as being selected as the source of TX_CLK_IN.
- **TX_CLK_OUT:** This is the signal selected to be provided to the UNAE main board as the transmit reference clock. In SONET mode, it is a buffered version of the TX_CLK_DIV2 signal (77.76 MHz), and in Gigabit Ethernet mode it is a buffered version of the TX_CLK_31 signal (31.25 MHZ).
- **RX_CLK_IN:** This is the receive clock provided as an input to the FPGA by the AMCC 3057. In a Gigabit Ethernet mode it will be 78.125MHz, and in a SONET mode it will be 155.52MHz. This clock is used to move data into the receive 16 to 32 bit converter, as well as providing the reference for both RX_CLK_DIV2 and RX_CLK_125.
- **RX_CLK_DIV2:** This signal is a divide-by-2 derivative of RX_CLK_IN. In a SONET (77.76MHz) mode it is selected as the source of the RX_CLK_OUT signal, and in a Gigabit Ethernet (39.0625MHz) mode it serves to clock data out of the 16-to-32 bit converter and into the receive FIFO.
- **RX_CLK_125:** This clock (125MHz) is an 8/5ths multiple of RX_CLK_IN, and is used only in Gigabit Ethernet mode. It provides clock to the Code Group Align module, The 10B/8B decoder module, the Receive Synchronization State Machine, as well as serving to clock decoded data bytes into the 8 bit to 32 bit converter, and as a reference for RX_CLK_31.
- **RX_CLK_31:** This clock is a divide-by-4 derivative of RX_CLK_125. Utilized only in a Gigabit Ethernet mode, it is used to clock data out of the 8 bit to 32 bit converter, and is the source for RX_CLK_OUT.
- **RX_CLK_OUT:** Driven to the UNAE main board as the receive synchronization source, this clock is created by a MUX selecting RX_CLK_31 when in Gigabit Ethernet mode, and RX_CLK_DIV2 when in SONET mode.

4.8.2.11 Test Approach

Testing in the lab began in earnest with the verification of the microprocessor accessed registers. Once basic read/write capability was verified, the parts were configured for

normal operation, and the clock synthesis features in the PMA-PCS FPGA were verified. This was a necessary first step, as these clocks served as the data path clocks for all other portions of the system. Upon verification of system clock generation and distribution, the testing moved on to data path verification.

Verifying the passage of packets through the data path was done, one section at a time, from the backplane out. This approach was used because a known data source – i.e. the SONET design - was available at this time. Using an IXIA IP tester to transmit packets to the SONET design, packets were transmitted across the backplane. Since the Gigabit Ethernet system implements the same CPE code as all other UNAE systems, the data path into the FIFO was already known to be good. Thus, the unknown data path was tested one portion at a time using loop backs. With each portion verified, the loopback could be stepped a bit further out from the FIFO. So the MAC FPGA data path was verified first, and then the communication between the MAC and the PMA-PCS FPGA was verified.

Once the data path was verified out to the point where PCS layer encapsulation begun, the testing approach was changed to working in towards the backplane from the optical interface. This approach was taken because it allowed a subset of the PCS layer functions to be verified first, and then slowly add to that subset. More specifically, the ability of the system to interoperate with a known Gigabit Ethernet interface in an IDLE only mode was tested first. This means testing the simplest form of interaction between to peers without any packets or configuration messages involved. Next, the ability to exchange packets was verified, and finally the ability to exchange configuration messages and perform Auto Negotiation was tested.

A major limitation in the debug and verification of the PCS layer characteristics was the lack of a piece of test equipment, which provided easy access to the bit-level information which comprises the PCS layer. This obstacle was overcome to some degree by the use of a Communication Signal Analyzer equipped with an optical interface and a serial trigger.

Results of system testing of Gigabit Ethernet and the functional blocks described here are provided in section 5.2.1.

4.9 All Optical Label Switched Router Interface

4.9.1 OLSR Interface design objectives

For the last nine months the University of California at Davis and Tektronix have been collaborating to develop a client interface for an optical label switching network (OLSN). The main function of the client interface is to provide an interface between the OLSN and a connecting network such as an IP over SONET network as shown in Figure 23. The ingress client interface is responsible for modifying the incoming packets to a format that is suitable for transmission through the OLSN. This includes adding the appropriate label with corresponding routing and control information. The egress client interface receives packets from the OLSN and converts them back into the format the connecting network needs. The OLSN is an unslotted, variable packet length network that uses labels to route

each packet to its destination. Due to optical switching in the OLSN, at each egress client interface, the clock needs to be recovered from the received data. As a result, this network is asynchronous and differs from a SONET network in that it does not have empty frames being sent when there is no data being transmitted.

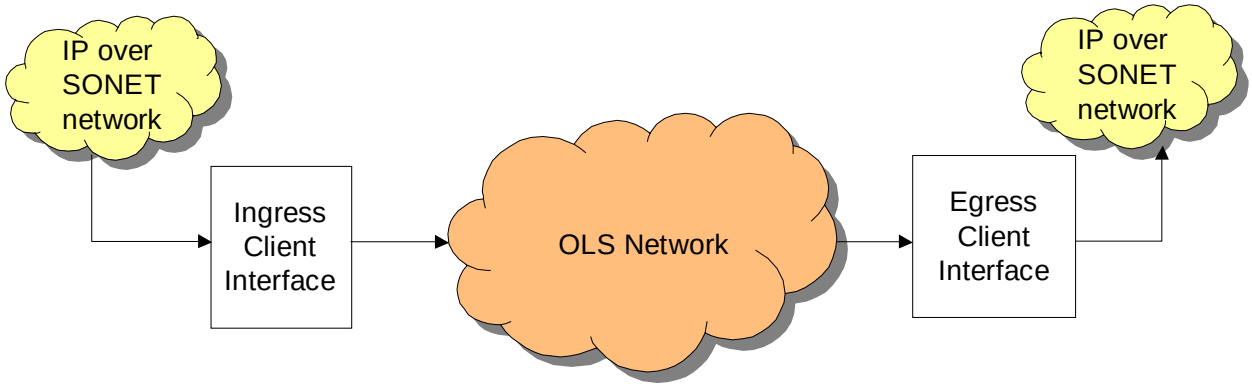


Figure 23 Overall network including all optical label switched and SONET network domains

The UNAS will be used as the ingress and egress client interfaces to the OLSN. The client network uses the packet over SONET (POS) protocol. Each UNAS will have two UNAEs, a POS UNAE and a UC Davis UNAE. A UNAE contains two types of boards, a main board and a mezzanine board. The POS UNAE and the UC DAVIS UNAE differ only by their mezzanine boards. The POS UNAE is capable of transmitting and receiving OC-48 POS. The UC Davis UNAE uses HDLC encapsulation to transmit and receive IP packets to and from the OLSN. The IP packet could be encapsulated using any method, but HDLC was chosen since it is a standard.

In the ingress client interface, the POS UNAE is used to receive and convert the POS signal into bare IP packets. The packets transverse the backplane data bus to the UC Davis UNAE where the IP packets are encapsulated into HDLC frames and an optical label is assembled. This path will be referred to as the ingress path.

In the egress client interface, the UC Davis UNAE is used to receive HDLC frames from the OLSN, decapsulate the IP packets, and send them to the backplane data bus. The POS UNAE pulls the bare IP packets from the backplane data bus, converts them into a POS signal, and sends them to the client network. This path will be referred to as the egress path. Figure 24 illustrates the ingress and egress paths.

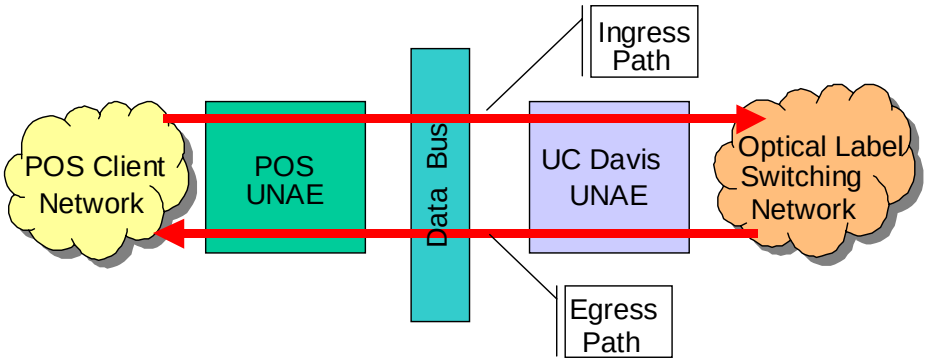


Figure 24. Ingress and Egress Paths of the Client Interface

Referring to Figure 4 the performance of the UNAE in a packet over SONET (POS) application has been covered in detail above in section 4. The POS card interfaces to either an OC-48 router or another source of POS to deliver bare IP packets to the backplane.

For the UC Davis UNAE, the lower data path in Figure 4 is the ingress path. The CPE is to receive IP packet data from the backplane data bus. It is to outputs the IP packet data to the FIFO. The PLPE fetches the data from the FIFO. The PLPE generates the label from the IP header data and encapsulates the data in HDLC. The data and label are then sent to the Framer FPGA. The framer scrambles the data and sends both the data and label to the mezzanine card where they are output to the OLSN. The framer also generates extra HDLC preamble bytes that are inserted at the head of the packet and are used by the egress UNAE to recover the clock from the data. The clock recovery chip on the egress mezzanine board specifies 250µs of data is required before the PLL locks and a clock is recovered. The extra preamble bytes could be eliminated if there was a 2.5Gbps burst mode receiver available. This is the role the BRAD was to have played.

The upper data path in Figure 4 is the egress path for the UNAE used by UC Davis. HDLC encapsulated IP packets are received from the OLSN. The Optical Router Client mezzanine card receives data that has already been converted from the optical domain to the electrical domain. It takes the data and extracts the clock using the preamble bytes. The data is passed to the framer FPGA where it is descrambled and byte aligned. The byte aligner is required since the OLSN is asynchronous. Consequently the bits are not byte aligned. The PLPE then extracts the HLDC frame and sends it to the CPE. Finally, the CPE sends the data to the POS UNAE over the backplane data bus.

Tektronix supplied a Platinum platform, controller module, four UNAE boards and four 2469XC optical mezzanine cards, and a full VxWorks build environment to UC Davis for this work. There are three main modifications to the FPGA code in the UNAE (illustrated in Figure 25) which UC Davis had to make to create a working 2.5Gbps Optical Router Client interface. These modifications enabled:

1. Generation of the label in the PLPE ingress path (green hatched)
2. Addition of a data multiplexer to include extra HDLC preamble bytes in the framer ingress path (blue hatched)
3. Addition of a byte aligner in the framer egress path (yellow)

In the next sections each of these three modifications will be discussed in detail.

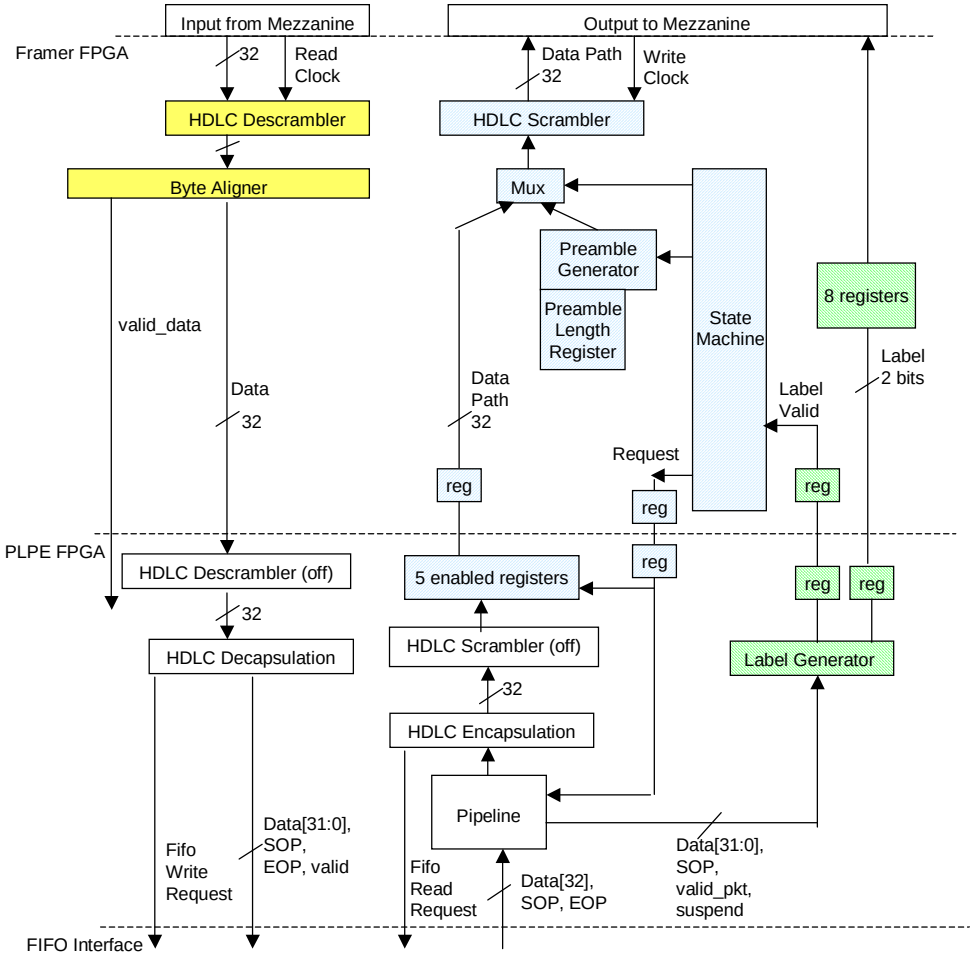


Figure 25. UC Davis UNAE Modifications

4.9.2 Summary of Work

4.9.2.1 Label generation and coding approach

The work on this portion of the project was done partly by UC Davis researchers and

partly by the Tektronix NGI team. For the data flow to the ingress side of the OLSR, the architecture and FPGA code developed for label generation, preamble generation, and framer scrambling was done by UC Davis as shown in Figure 25. They also implemented the data descrambler, the frame synchronization/Byte aligner in Figure 25 for the egress side of the OLSR. Tektronix engineers provided consulting and some working code elements that could be modified for the blocks above.

Figure 26 shows the inputs and outputs of the label generation module.

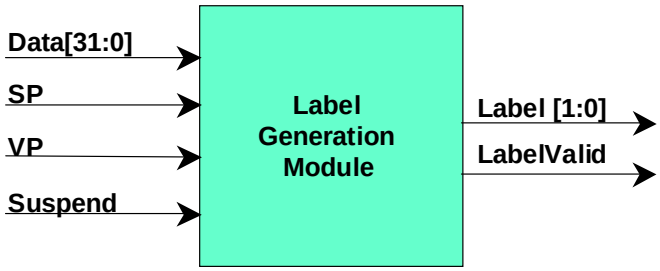


Figure 26 Input and Outputs of the Label Generation Module

There are four inputs: *Data*, *SP*, *VP*, and *Suspend*, which come from the data pipeline in the PLPE FPGA. *Data* is the next 32-bits of the IP packet, *SP* is active when a start of packet occurs, *VP* is active when there is valid data on the 32 data lines, and *Suspend* is active when the data pipeline is suspended and therefore not sending data out. The module takes the information in the IP packet header and makes a label useful to the OLSN optical interface. The first output of the module is a 2-bit *label*. Two bits are needed since the internal clock rate of the FPGA is 78 MHz and the final label output needs to be double the clock rate, 155Mbps. The second output is the *LabelValid* signal that is active during the first clock cycle of a valid label. The label is made from information in the IP header of each IP packet.

Figure 27 shows a color-coded IP header and the label that is derived from each colored or patterned field of the header.

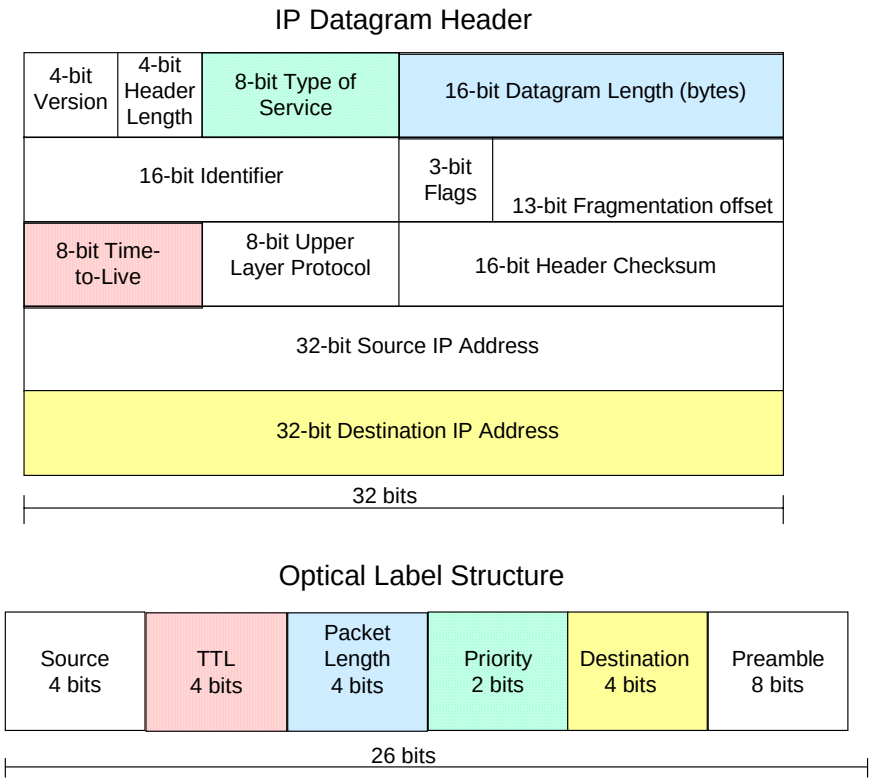


Figure 27 IP Header and Label Structure

The fields of the label are the following:

- *Preamble:* A unique 8-bit sequence. It is used to detect the start of a valid label. Each core router in the OLSN detects this unique 8-bit sequence to indicate the arrival of a new packet and uses the remaining fields of the label to intelligently switch the new packet. This field is constant for all routers.
- *Destination:* 4-bit destination address of the new packet. This is the address of the egress client interface. The core routers use this field to perform the routing table look up and switch the packet to the correct output. The ingress client interface generates this address by looking at the 32-bit destination IP address of the IP header.
- *Priority:* The priority of the corresponding packet. This field allows priority based routing, which is an essential feature to provide differentiated classes of service. The ingress client interface takes the 8-bit type-of-service field in the IP header and coverts it into a 2-bit priority.
- *TTL:* Similar to IP (Internet protocol) the 4-bit TTL or time-to-live field represents the maximum number of routers a packet may switch through before it is dropped. The TTL is effectively a hop count limit on how far a packet can

propagate through the network. The ingress client interface assigns this 4-bit value by looking at the 8-bit TTL value in the IP header.

- *Packet Length:* This field contains the 4-bit encoded length of each incoming packet. The ingress client interface assigns this value by looking at the 16-bit datagram length field in the IP header.
- *Source:* 4-bit source address of each packet. This is the address of the ingress client interface and will be constant for each client interface.

4.9.2.2 Label Generation Module

Figure 28 shows the architecture of the label generation module. The architecture is designed in a manner that allows easy changes and expansions. There are seven code units that make up the label generation module. These are the input register unit, the input enable state machine, the output register/output enable state machine, and four code blocks for calculating priority, length, time to live and destination address label elements.

The first unit is the input register unit. All input registers are connected to some portion of the 32-bit *Data* input. These registers are:

Register	Length	Function	Connected to data lines
I_tos	8 bits	IP header type-of-service	23-16
I_length	16 bits	Datagram length in bytes	15-0
I_ttl	8 bits	IP header time-to-live	31-24
I_dest	32 bits	Destination address	31-0

The input registers are write enabled by the second unit, the input enable state machine (IUSM). Details of the state machine are provided in the "UC Davis Subcontract Final Report. The state machine starts when both *SP* (start packet) and *VP* (valid packet) are active. Using the three input signals the IUSM steps through the full IP header, extracting the appropriate header bits into the input registers. When the final state of the IUSM is entered, the state machine waits for the PLPE pipeline to start sending data. This is indicated by a low value on the *Suspend* line. Once data is being sent, the *start* signal is asserted during the next transaction. The *start* signal tells the output unit to start outputting the label.

Following this, the IUSM either enters a wait state for SP and VP to be asserted again; or if SP and VP were both high during final state S7, the next label is processed.

The third and fourth code units in Figure 28 are the output registers and output enable state machine (OESM). There are six registers, one for each field in the label. Two of the registers are constant, the preamble and the source. The label is output two bits at a time.

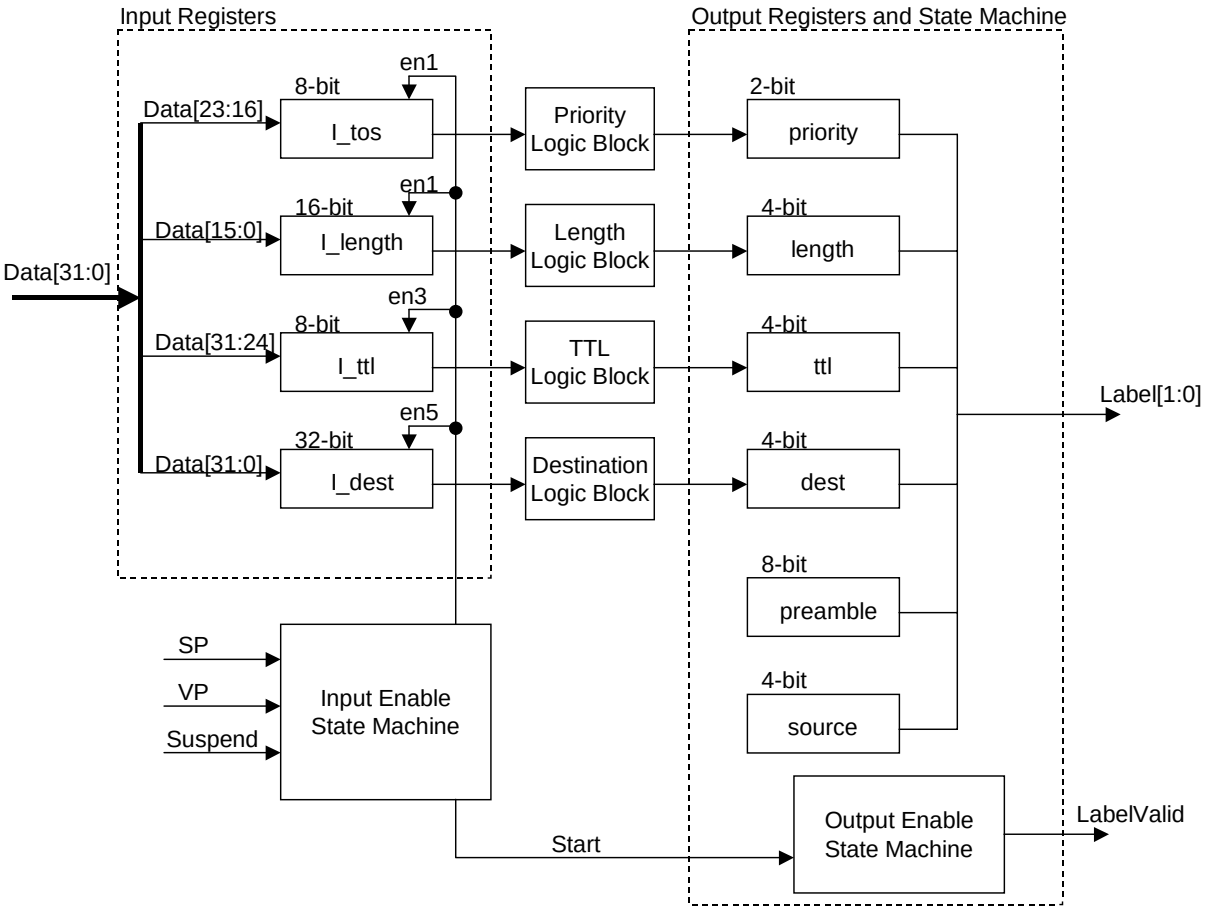


Figure 28 Architecture of the Label Generation Module

Hence, the 26-bit label is output in 13 clock cycles. The machine waits in state S1 for the *start* signal from the input enable state machine to be asserted. Also in S1, the constant registers, preamble and source, are initialized. Once the *start* signal is asserted, the 8-bit preamble register is output in states S2, S11, S3, and S4. The *LabelValid* signal is asserted in state S11 to indicate the start of the label. The label output is registered while the *LabelValid* signal is combinational. Therefore, the first cycle the label appears is during state S11. After the four clock cycles for the preamble, the 4-bit destination register is output. Next, the 2-bit priority register is sent, followed by the 4-bit ttl register, the 4-bit length register and finally the source register. This completes the label. In states S15 and S16 the label output is zeros; in states S17 and S18 the label output is ones. These four bit-stuffing states are required by the hardware in the core router. The state machine loops in these four states until the start signal is asserted. It then proceeds to output the next label. Details of the output enable state machine were detailed in the UC Davis Subcontract Final Report.

In Figure 28, the last four units are logic blocks that compute the label output register values. The blocks are as follows:

- *Priority Logic:* This block maps the 8-bit *I_tos* register to a 2-bit *priority*.
- *Length Logic*: This block maps the 16-bit *I_length* register and coverts it into a 4-bit *length*. Datagrams are rarely greater than 1500 bytes. For this reason a nonlinear mapping of the length bits will be used. For lengths less than 128 bytes the value 0001 is assigned to the *length* register so the *length* field is not zero for small packets.
- *TTL Logic:* This block converts the 8-bit *I_ttl* register into a 4-bit *TTL*.
- *Destination Logic:* The 32-bit *I_dest* register is mapped into a 4-bit destination address, that of the egress client interface. Ideally, the mapping would be done in a look-up-table updated by the network management and control. Currently, the last 4 bits of the IP address are being used.

The above four logic blocks were implemented as separate code units so they could easily be changed without affecting other logic blocks.

The framer FPGA was also modified for the label generation. As shown in Figure 25, eight serial registers were added to the label path to compensate for the delay the data payload experiences. This is done so the label and data payload exit the UNAS at the same time.

4.9.2.3 Framer Ingress Path Data Mux

The ingress path in the framer FPGA (ingress client interface) contains a module that controls the data output as shown in Figure 25. The data is either HDLC preamble bits or the HDLC encapsulated IP packets. Recall that HDLC preamble bits are required since there is no access to a 2.5Gbps burst mode receiver. Therefore, at least 250 msec of extra preamble must be generated so the egress UNAS can recover the clock from the received data. Figure 29 shows the block diagram of this module.

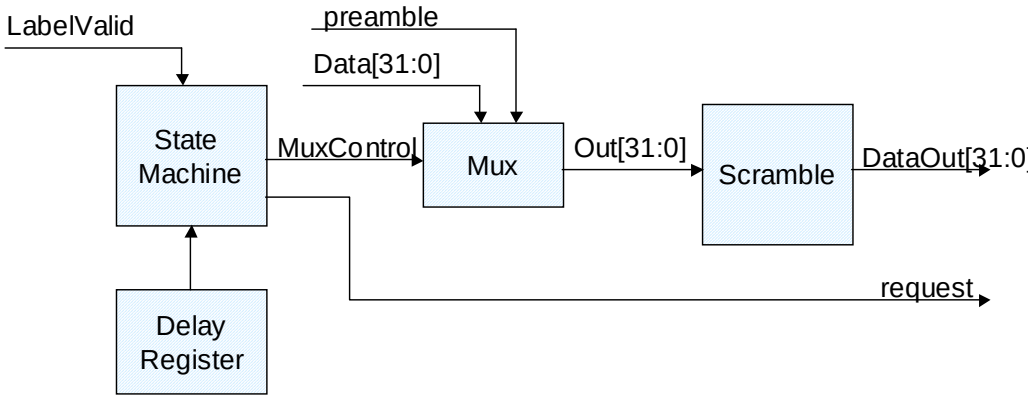


Figure 29 Block Diagram of Framer Egress Path Data Mux

The module consists of three code units and a register. The Mux code unit accepts data and HDLC preamble bits at its inputs. When the *MuxControl* line is low, data is sent to the *Out* lines. When *MuxControl* is high, the HDLC preamble byte, 01111110, is sent repetitively to the output. The output of the mux is fed into the scrambler code unit that performs the HDLC data scrambling. The scrambler is the same parallel self-synchronous $X^{43} + 1$ scrambler previously described.

The state machine controlling the mux has the following inputs and outputs.

- *LabelValid:* Input signal from the PLPE label generation unit that indicates the first clock cycle of a valid label.
- *Delay*: 16-bit input signal from the PCI registers. This register contains the number of clock cycles during which HDLC preamble bits should be sent.
- *MuxControl*: Output signal that controls the mux. When low, data is output. When high, HDLC preamble bits are output.
- *request*: Output signal going to the PLPE data pipeline (see Figure 25) to request data. When high, data is being requested. When low, no data is sent.

The state machine assures that there is valid data requested from the PLPE and flowing through the mux to the output whenever there is a valid label to accompany the data. Between times when there is a valid label, a preset number of preamble bytes is sent. The length of preamble time is initialized before the preambles are needed.

The PLPE FPGA was also modified for the data mux. As shown in Figure 25, five *request* enabled registers were added to the output of the PLPE data pipeline. The registers are an extension of the PLPE pipeline that are needed to meet timing requirements between the label generation unit and the data mux unit.

At the egress client interface, the HDLC encapsulated IP packets are received. Since the UC Davis network is asynchronous, there is no knowledge of where byte boundaries occur when packets are received. Therefore, a byte aligner is needed. The byte aligner takes the incoming HDLC frame and aligns it according to the preamble bits, 01111110. Figure 30 shows the block diagram of the framer egress path module. First, the data is descrambled by the self-synchronous $X^{43} + 1$ descrambler. The data is then passed to the byte aligner unit. The byte aligner unit aligns the input data to byte boundaries and outputs byte-aligned data, *aligned_data[31:0]* and a signal indicating when the aligned data is valid, *valid_data*. The *lock_det* signal from the mezzanine board is an input to the byte aligner that indicates when the clock has been recovered from the data and there is a bit-lock on the data. The *pre_count* register is an input from the PCI register block. It designates how many preambles to detect before a byte alignment is chosen.

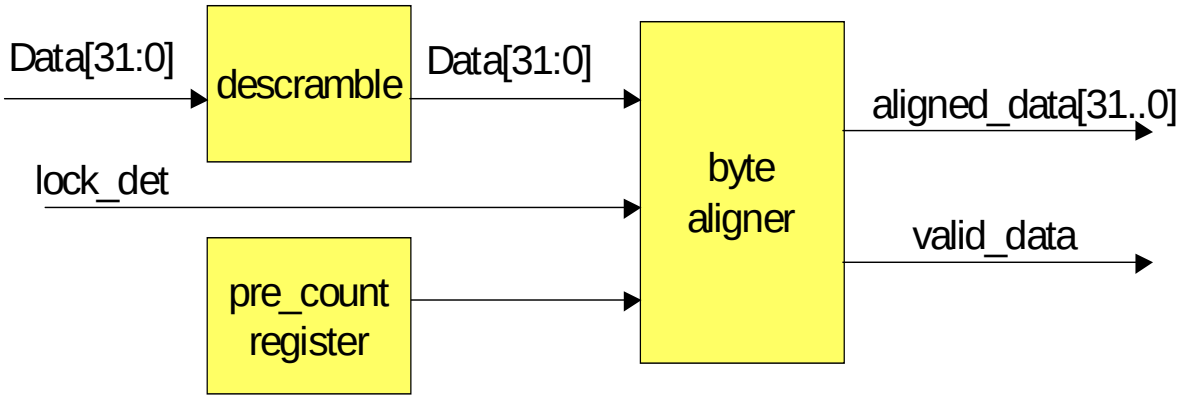


Figure 30. Block Diagram of the Framer Egress Path

The byte aligner has several code units. Detailed diagrams and explanations of signals and processes through this unit are available in the UC Davis Subcontract Final Report.

5 Results and Conclusions

5.1 System Software and Real Time Operating System

The software for the NGI program was divided into two separate tasks: The Windows User Interface with Windows driver system and the VxWorks real time operating system. This design proved to be beneficial in several ways, with few issues. The VxWorks portion could be compiled and updated independently of the Windows UI, allowing the hardware and firmware team to make changes rapidly. Another benefit was that VxWorks code was developed before the first hardware cards were available, allowing testing of the first hardware prototypes within days of receiving them. A drawback to this approach was keeping the system in sync. Since there is a common messaging thread to connect the VxWorks and the Windows UI, any change in the overall feature set required the messaging system to be redefined. Once the updated messaging system was redefined, all major components, including the VxWorks application, the NT driver, and the Windows UI, were recompiled to keep them in sync. The system architecture proved to be robust and flexible, even with residual code management issues. The NGI team would choose the same approach again.

The remote UI worked very well: With initial testing, there were not any observable speed issues or connection problems. The design of a Java remote UI interface uncovered some surprising issues, such as time to load and a crude appearance. The team chose the Win32 approach - mainly because of the availability of several Windows PCs, allowing the team to solve multiple technical issues at once.

The VxWorks “shell” connected to the Windows UI proved very useful. The user did not have to connect a separate terminal to the NGI hardware to gain access to the VxWorks shell, as with other products using VxWorks. In fact, it worked very well: The hardware team preferred to use the shell for almost all hardware debug, rather than use the UI for setting up the instrument. During the program, we discovered the need for using a “hardwired” terminal. The VxWorks shell depends on everything in the system working correctly, from the Windows UI, Windows driver, VxWorks, and hardware since the VxWorks shell was displayed through the UI. The NGI team developed an RS232 interface to allow testing of the VxWorks during the initialization stage, where the Windows system was not available. This proved to be an important part of the program.

The PPP negotiation software worked very well. In testing of the PPP software, it was discovered there were differences between Cisco and Juniper OC48 routers. The PPP negotiation worked exactly the same in the LCP portion of the negotiation, but differently in the IPCP portion of the negotiation. The area where it worked differently was in the IPCP request for the IP address. With a Cisco 12000 GSR and with an Ixia test set, they responded with their IP address. The Juniper router, however, would ignore status requests, and essentially loop forever: making a request, waiting for a response, and not getting one, making a new request, etc. This was fixed by modifying our software to wait a number of times for the response, as the NGI does not use this information. However,

this is a customization that does not conform to the PPP standard.

The auto negotiation for Gigabit Ethernet was very straight forward and was working within days of the first FPGA implementation. It was easier to implement than the PPP negotiation and has been working flawlessly after a few FPGA turns.

In conclusion, the Software Architecture taken was the right approach. It was straightforward to partition the tasks, a solid and proven design, it met the needs of the program, and it satisfied the requirements of the hardware and software engineers on the program

5.2 System testing and performance

5.2.1 Testing of packet and video interfaces

5.2.1.1 Single board testing

Packet over SONET packet tests

Initial testing of the UNAS was done with individual boards. Loopbacks were designed in the hardware at various points in the data flow to enable testing of individual components. For UNAE boards configured as OC-48 Packet-over-SONET, loopbacks were possible at the following locations: 1) at the serializer/deserializer chip on the optical mezzanine card, and 2) at the CPE interface to the hardware FIFOs (RFIFO and TFIFO). Internal error monitoring is done in both transmit and receive directions in the Framer (SONET diagnostics) and the PLPE (packet diagnostics). Packet testing was done to verify each board's hardware operation, since the full 32-bit width of the data path is exercised. Sixteen UNAE boards were built, and fourteen of those were verified as having no errors during packet loopback testing.

Gigabit Ethernet packet tests

The results of the testing proved that the basic features of Gigabit Ethernet were successfully implemented, with some minor issue. The design has the ability to exchange IP packets at or near the theoretical maximum rate for a given packet size and inter-packet gap rate. This means approximately 960Mbits of user (IP packets) traffic can be exchanged without any detectable corruption of the contents of the MAC frames. The Auto negotiation feature was also verified to perform the basic link negotiation tasks necessary to interoperate with like equipment. Both the ability to pass IP traffic and to Auto Negotiate were also verified with a Cisco router by passing uncompressed standard definition video signal in both directions.

There are several limitations to the design as it stands. First, the design cannot talk to itself or run in a loopback (towards the backplane) mode due to the limitations of the Crystal Oscillator used as a local clock source. The design must use the clock recovered from the line, thus acting as a slave to its peer interface. If configured to run in local time,

it will take running disparity errors and code group errors at the peer interface. Second, the statistics gathering that is presented on the GUI counts what appear to be false errors at a fairly low rate. These errors are shown as received code group, as well as packet length errors. These all appear to be false error reports, as they are never reflected in the data integrity. Finally there remains one issue with Receive PCS portion of the PMA-PCS FPGA: Occasionally, when the Auto Negotiation software changes the XMIT variable (which controls whether all idle codes, configuration messages or packets are transmitted), the design goes into a failure mode which the system reports as line rate FCS errors.

No extensive network characterization, such as delay variation insertion, has been performed on this system.

Video tests

Similar loopback testing of UNAE boards configured as video boards was also done. The video configuration differs from the packet configuration in having a different mezzanine interface (SD or HD video) and different code in the Framer and PLPE FPGAs. For video boards, loopbacks are possible at the following locations: 1) in the Framer FPGA through an internal rate-matching FIFO, and 2) at the CPE interface to the hardware FIFOs. All boards verified by packet loopback testing were also verified by HD video loopback testing.

5.2.1.2 Multiple board testing

Video to POS/GbE

Testing of packet transport across the backplane of the CPCI chassis can be accomplished by configuring a video card (Link card) in one slot, and configuring a second card as a POS or GbE (Phy card) in the next higher-numbered slot. When video transport (standard definition for GbE, either standard definition or high definition for OC-48 POS) is turned on in video card, packets containing video information are sent across the backplane to the Phy card. Configuring the CPE in the Phy card to “Loop” state will return the packets to the video card for decoding; this allows testing of CPCI backplane transport. Configuring the CPE in the Phy card to “Phy” will send the packets through the card and out the optics on the mezzanine in whichever physical layer format is selected. Packets can then be examined or monitored using a Gigabit Ethernet packet analyzer or a POS packet analyzer.

Shown in Figure 31 below is an analysis of packet inter-arrival time for HD video packets captured by an Adtech AX4000 OC-48 packet analyzer. The difference between maximum and minimum inter-arrival time is 1.57 microseconds, indicating a very uniform packet flow. Shown for comparison purposes in Figure 32 is a constant rate packet flow generated by the packet analyzer for the same size packets and similar data rate; the spread in inter-arrival times for this flow is 2.10 microseconds.

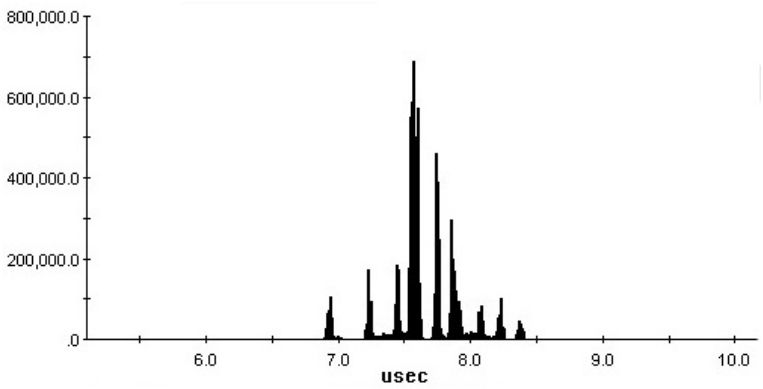


Figure 31 Packet inter-arrival time variation for 1464-byte HD video packets generated by a Video UNAE and output as OC-48 POS by a POS UNAE.

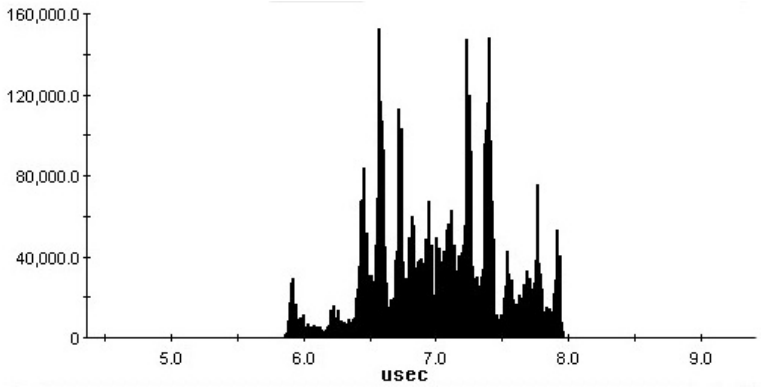


Figure 32 Packet inter-arrival time variation for 1464-byte POS packets generated at a constant 1.6 Gb/s rate by an Adtech AX4000 packet tester

Video packets exiting the Phy card optical port can also be looped back into the receive optical port using a fiber patch cord and sent back to the video card for decoding. In order for this to operate correctly, however, the transmit port on the Phy card mezzanine must be set to "local time" to use a locally generated clock. Otherwise the transmitter attempts to use the recovered receive clock ("loop time"), and frequency can drift substantially.

Note: It was found that the clock oscillators (155.52 MHz for POS, 156.25 MHz for GbE) built onto the third revision of the optical mezzanine had too much jitter to operate without errors when the UNAS boards were used in the "local time" stand-alone mode as described above. Some of the new mezzanine boards had the POS clock oscillator replaced with a previous version of the oscillator with lower jitter. All of the mezzanines will operate correctly when connected to a router or switch and used in "loop time" mode.

POS to GbE

The UNAE cards can also be set up to act as a physical layer translator between to

protocols. Using two cards adjacent to each other in the CPCI chassis, the card in the lower-numbered slot is configured as Link and the card in the higher-numbered slot is configured as Phy. Packets received on the card configured as Link (an OC-48 POS card, for example) will be sent to the second card configured as Phy (a Gigabit Ethernet card, for example) and exit the transmit port as Gigabit Ethernet packets. Packets generated by a POS packet analyzer can then be captured by a Gigabit Ethernet packet analyzer and compared for errors.

Error-free operation of POS to GbE and GbE to POS was verified using packets generated by a packet analyzer in one format and analyzed in the other format.

Video to video (multiple chassis)

Packets generated from video input using a Video/POS or Video/GbE card pair as described in the first section above can be routed to a second card pair of the same type using a length of optical fiber. Packets will be transmitted in whichever format is selected for the Phy port. This configuration is logically equivalent to the fiber-loopback described above, and is subject to the same constraint: the transmit clock must be set to "local_time", since there is no receive clock to recover. Video transmission can be done through an arbitrary length of single-mode fiber, subject to the link power budget of the transmitter/receiver pair used (13 dB or approximately 30 km for the transceiver used on the UNAE optical mezzanine).

Direct chassis-to-chassis connections were done for both standard definition video and high definition video in OC-48 POS packets. Both operated for extended periods (greater than 12 hours) with no errors.

5.2.1.3 Testing through router ports

POS to POS

Pairs of UNAE cards configured as described above in multiple card testing (Video/POS) can be used to inject packets at one router port and remove them through another. Testing was done for HD video in packets between two OC-48 POS router ports on a Cisco GSR 12008 router. PPP negotiation between the UNAS and the router established the IP address associated with each port.

Shown in Figure 33 below is a histogram of packet inter-arrival time for HD video POS packets generated by the UNAS (Video/POS board pair), input to a Cisco router OC-48 POS port, and routed to a second OC-48 POS port for output. The difference between maximum and minimum packet inter-arrival time is 5.40 microseconds, indicating that the router alters the distribution of packet arrival times. The change in inter-arrival time is minimal for a single router with no congestion, but can become a problem with many routers in a congested network (see section 5.3).

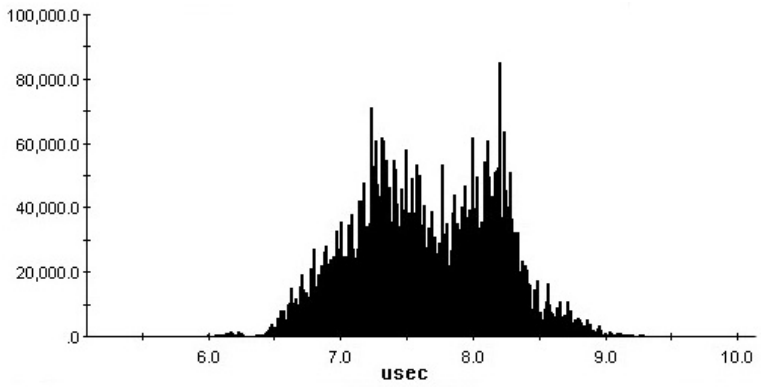


Figure 33 Packet inter-arrival time variation for 1464-byte HD video in POS packets generated by the UNAS and routed between Cisco GSR OC-48 POS ports.

GbE to POS

Video packets can also be injected into a GbE router port (for standard definition video only; the data rate of uncompressed HD video at 1.5 Gb/s is too high for GbE) and recovered from a POS router port, or vice versa. Lengthy tests were run of standard definition video routed between GbE and OC-48 POS ports on a Cisco GSR 12008 router with no errors. Packet inter-arrival time measurements were also made for standard definition video packets that entered the router through a GbE port and exited through an OC-48 POS port. The average inter-arrival time was 15.25 microseconds, but the difference between maximum and minimum times was more than 140 microseconds due to a small number of large gaps.

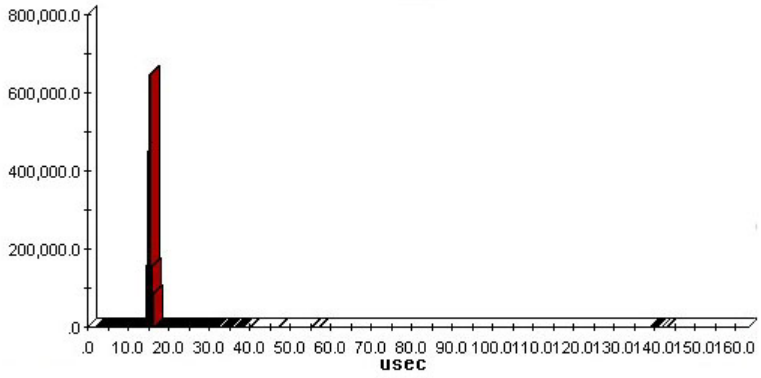


Figure 34 Packet inter-arrival time variation for 564-byte SD video in IP packets generated by the UNAS and routed between Cisco GSR GbE and OC-48 POS ports.

5.2.2 Wide area network experiments

5.2.2.1 University of Washington (Seattle) to ISI East (Wash., D.C.)

Initial testing of uncompressed high definition video transport over a wide area packet network was done over a link between the University of Washington in Seattle and the

University of Southern California Information Sciences Institute in Washington, D.C. Testing over the wide area network began on 11/1/2001 using the Abilene network (Figure 35). Initial testing was done using a route through Denver with 11 router hops (Seattle to Denver to Kansas City to Indianapolis to Cleveland to New York to Washington, D.C., with multiple routers in the start and end cities).

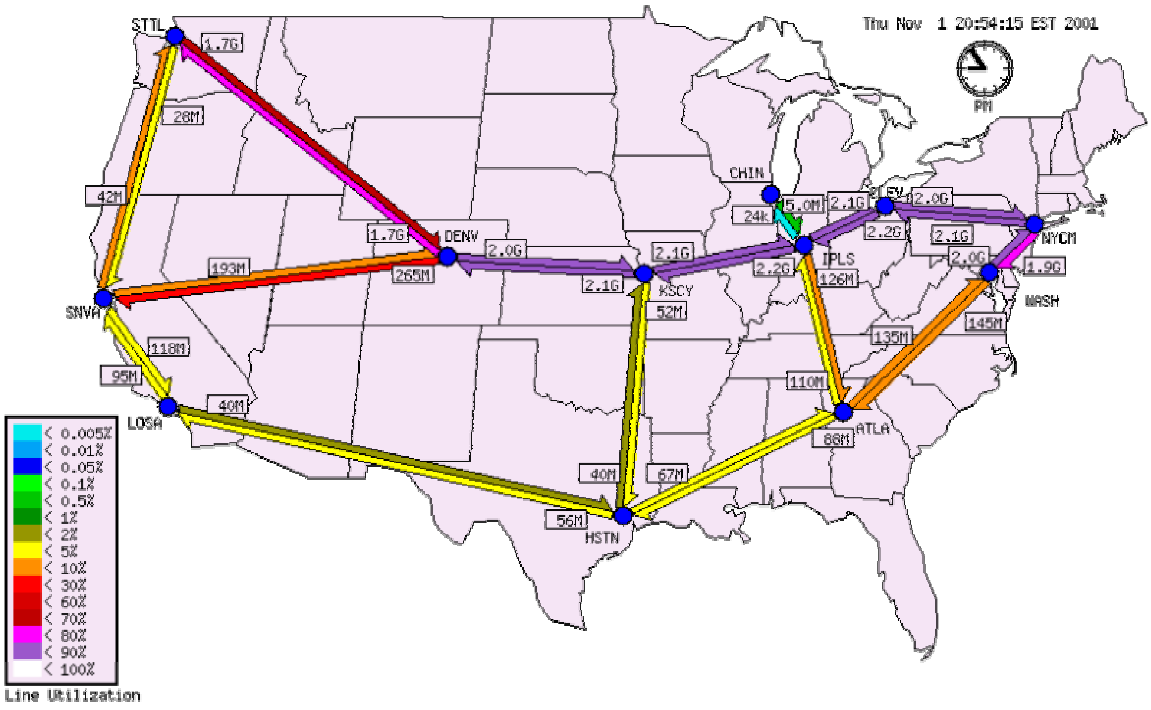


Figure 35 Network utilization during testing of HD video transport over the Abilene OC-48 Packet-over-SONET network.

The first tests used a default packet size of 564 bytes, and several thousand packet sequence errors per second were observed (out of a total packet rate of 355,438 packets/second). Video was observed on test monitors in Washington, D.C., but there were many glitches in the picture. By increasing the packet size used to carry video (and thereby decreasing the packet transmission rate), it was possible to obtain error-free video over the northern route shown in Figure n. Measurements of the distribution of sequence number errors were made and are reported in Table 6.

Table 6 Sequence errors for different sizes of packets transporting HD video over the Abilene network from Seattle to Washington, D.C. via New York on Nov. 6, 2001.

Sequence number difference	Rates of occurrence of sequence number differences (sec^{-1})		
	564-byte packets (355,438 pkts/sec)	1464-byte packets (130,159 pkts/sec)	4444-byte packets (42,006 pkts/sec)
≥ +5	68	0	0
+4	18	0	0
+3	451	0	0
+2	1,393	0	0
+1	352,353	130,159	42,006
0	0	0	0
-1	923	0	0
-2	158	0	0
-3	74	0	0

Additional testing was done over other wide area network segments. A second route tested was using the Abilene network from Seattle to Denver to Sunnyvale to Los Angeles, then the HSCC network over Qwest fiber from Los Angeles to Washington, D.C. The video data stream showed a loss of 12,866 packets/sec., 36% of the original data rate of 355,432 packets/sec. No video pictures were visible on test monitors with this loss rate.

The route above was modified slightly to avoid a congested link; the HSCC part of the network was routed through Atlanta. The received packet rates now showed that no packets were being dropped, but there were still substantial numbers of out-of-order packets observed even at the largest packet sizes used (Table 7).

Table 7 Sequence errors for different sizes of packets transporting HD video over the HSCC network between Seattle and Washington, D.C. via Atlanta on Nov. 8, 2001

Sequence number difference	Rates of occurrence of sequence number differences (sec^{-1})		
	564-byte packets (355,438 pkts/sec)	1464-byte packets (130,161 pkts/sec)	4444-byte packets (42,007 pkts/sec)
≥ +5	28,448	?	1
+4	12,692	?	4
+3	24,947	?	8
+2	55,310	?	94
+1	165,990	113,457	41,844
0	0	?	0
-1	37,539	?	45
-2	20,218	?	8
-3	10,294	?	3

The best explanation we currently have for the mis-ordering of packets through the network is that under certain condtions (high packet rates and high loading) the core network routers can change the ordering of packets because there are multiple paths through the routing fabric. When the time available to route packets is small (high packet rates) small delays in one of the paths can result in a change in the order of the packets. This effect was readily observable at the packet rates used for transport of uncompressed high definition video, but could be worked around in some cases by going to sufficiently large packets (sufficiently small packet rates).

5.2.2.2 Univ. of Washington (Seattle) to SC2001 (Denver)

The SC2001 Conference in Denver provided another opportunity for testing high speed data transport across a wide area network. A network was set up between the University of Washington in Seattle and the National Coordination Office for Information Technology Research and Development (NCO/ITR&D booth) on the SC2001 Conference show floor in Denver. A Level3 Packet-over-SONET network was used. There were at least three routers in the network: a Juniper M20 router in Seattle, a Juniper M160 router at the entrance to the SC2001 show floor, and a Juniper M20 router at the booth. The network path was unspecified, but probably went through northern California and several SONET repeaters and add/drop muxes.

Error-free uncompressed high definition video transport in IP/UDP/RTP packets was easily obtained. Even at a packet size of 564 bytes (355,435 pkts/sec stream) only a few mis-ordered packets were observed (Table 8). The pattern of sequence number differences for a packet size of 564 bytes show that the mis-ordering events are single packets skipped and then re-inserted one packet later; one out-of-order packet would produce three sequence number errors of +2, -1, +2 for this event which is what is observed.

Significant mis-ordering of packets was not seen on this network until the packet size was reduced to 244 bytes (924,145 pkts/sec stream). Even at this packet rate, an analysis of the pattern of sequence errors shows that approximately 99.5% of the mis-ordering events are single packets out of order by one place.

Subsequent to this test the capability to correct for single out-of-order packet events was added to the UNAS (see section 4.11.5). It has not been possible to test this capability in transmission experiments on a wide area network.

Table 8 Sequence errors for different sizes of packets transporting HD video over a Level3 Network between Seattle and Denver on Nov. 13, 2001.

Sequence number difference	Rates of occurrence of sequence number differences (sec^{-1})			
	244-byte packets (924,145 pkts/sec)	564-byte packets (355,435 pkts/sec)	1464-byte packets (130,161 pkts/sec)	4444-byte packets (42,007 pkts/sec)
≥ +5	0	0	0	0
+4	0	0	0	0
+3	37	0	0	0
+2	8,802	2	0	0
+1	910,879	355,432	130,161	42,007
0	0	0	0	0
-1	4,405	1	0	0
-2	22	0	0	0
-3	0	0	0	0

5.3 Results , experiments and conclusions for UNAS/OLSR interface experiments

All elements developed for the OLSR interface were shown to be functional. Although longer than expected preambles were required to get low jitter eye diagrams at 2.5Gbps, each element performed its function as designed. The system was tested out in the

proposed application, that of connecting a packet over SONET source with an all optical packet switched network using subcarrier multiplexing techniques to apply labels for directing the router's port mapping. The labels were first generated in the UNAS from information in IP packet headers.

Figure 36 shows the configuration of the UNAS for the experiments. Since there is only one UNAS mainframe at UC Davis, two client interfaces share the same mainframe. The ingress client interface receives POS traffic from the IXIA traffic generator. It then outputs the data payload and optical label to the OLSN. The egress client interface receives the data payload from the OLSN and converts it back into POS frames. The frames are then sent back to the IXIA traffic generator for analysis.

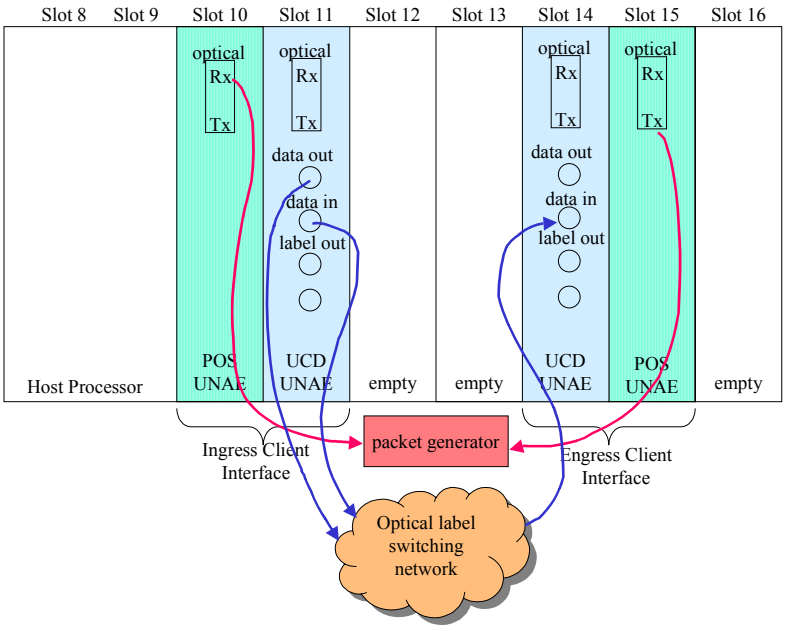


Figure 36 UNAS Configuration for Experiments

After the UNAS outputs the data payload and label, it enters the subcarrier-multiplexing transmitter. Figure 37 shows the details of the subcarrier-multiplexing transmitter (SCM TX) and voltage level adjustments. The SCM module and core OLSR had already been

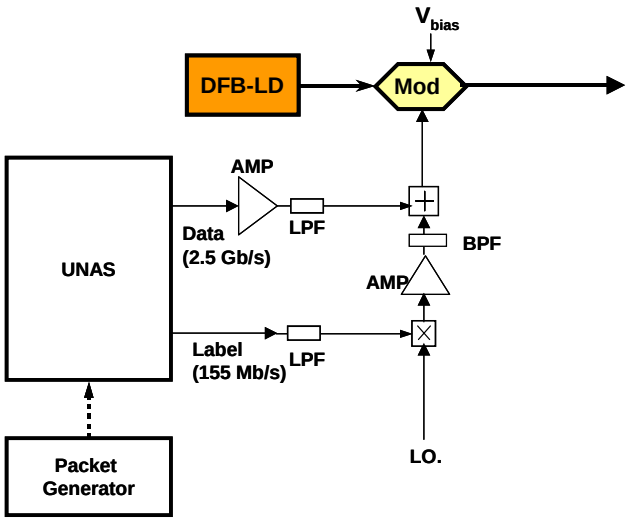


Figure 37 The detail of the SCM TX , (LO; local oscillator, LPF; Low-pass filter, AMP; Amplifier, BPF; Band-pass filter, Mod; Modulator)developed by UC Davis outside of this program.

The transmitter modulates the original label at 155 Mb/s on a 14 GHz carrier frequency subcarrier multiplexed with the baseband data payload packet at 2.5 Gb/s. The modulated signal will then include double-side-band subcarrier header appearing 14 GHz away from the center optical frequency.

Figure 38 shows the eye diagram of the electrical data payload from the UNAS. The bit rate of the data payload is 2.5 Gb/s. Figure 39 shows the bits of the label from the UNAS. The bit rate of the label is 155 Mb/s. These two figures show the UNAE as modified by UC Davis provides the data and labels to the input of the SCM transmitter. The SCM TX generates the optical packets containing the optical label (compressed header information) and payload, using double side-band sub-carrier multiplexing (SCM).

5.3.1 Optical Label Switching with UNAS

Figure 40 shows a functional block diagram of the experimental system, which represents the core of an Optical Label Switching (OLS) Router. It consists of an optical-SCM Tx, an optical-label/data separator, a label detector, a forwarding table, a switch controller, a tunable wavelength converter including a tunable laser (TL) and a Mach Zehnder Interferometer (MZI) semiconductor optical amplifier (SOA), a uniform-loss-cyclic frequency (ULCF) AWGR, and receivers. The optical-label/data separator consists of an optical circulator (OC1) and a Fiber Bragg Grating (FBG) with its peak reflectivity centered at the same optical frequency. The Burst Mode Receiver (BM Rx) receives the header signal and sends it to the Field Programmable Gate Array (FPGA), which is the core router control logic. According to the header content, the FPGA sends control signals to the TL driver to switch the wavelength. The TL output reaches the MZI Wavelength Converter (WC) and converts the payload signal onto the desired wavelength by cross-phase modulation. Payloads with different headers will be converted onto

different wavelengths. Thus they will go to different output ports of the AWGR. In this experiment packets with two different labels are transmitted. For simplicity, the two labels are alternating as $L_1\ L_2\ L_1\ L_2\ldots$ Packets with one type of label will be dropped, while those with the other type will be received and sent back to the UNAS. In the experiment both situations were tested: L_1 is dropped; or L_2 is dropped. The Client Interface consists of the elements shown in Figure 36 and Figure 38, shaded for clarity.

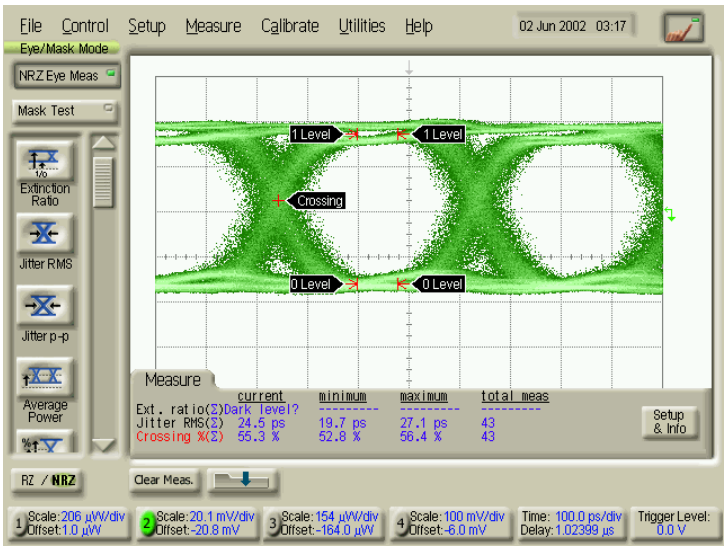


Figure 38. Eye diagram of the data payload from the UNAS (100 ps/div)

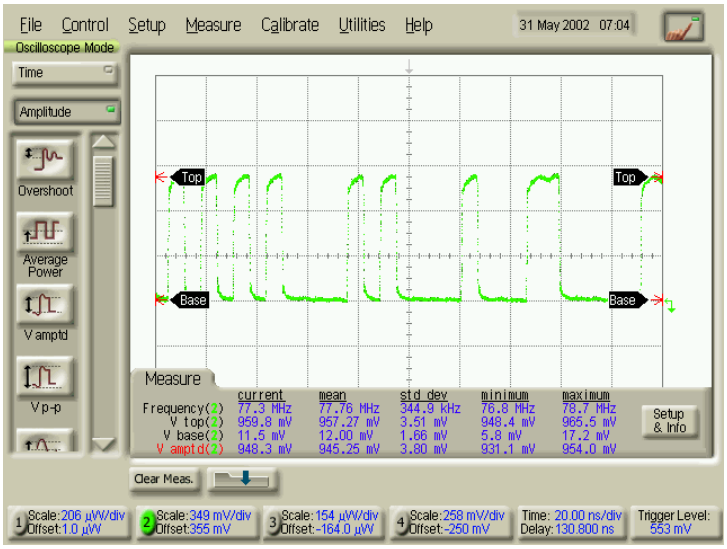


Figure 39. Bit trains of the label from UNAS (20 ns/div)

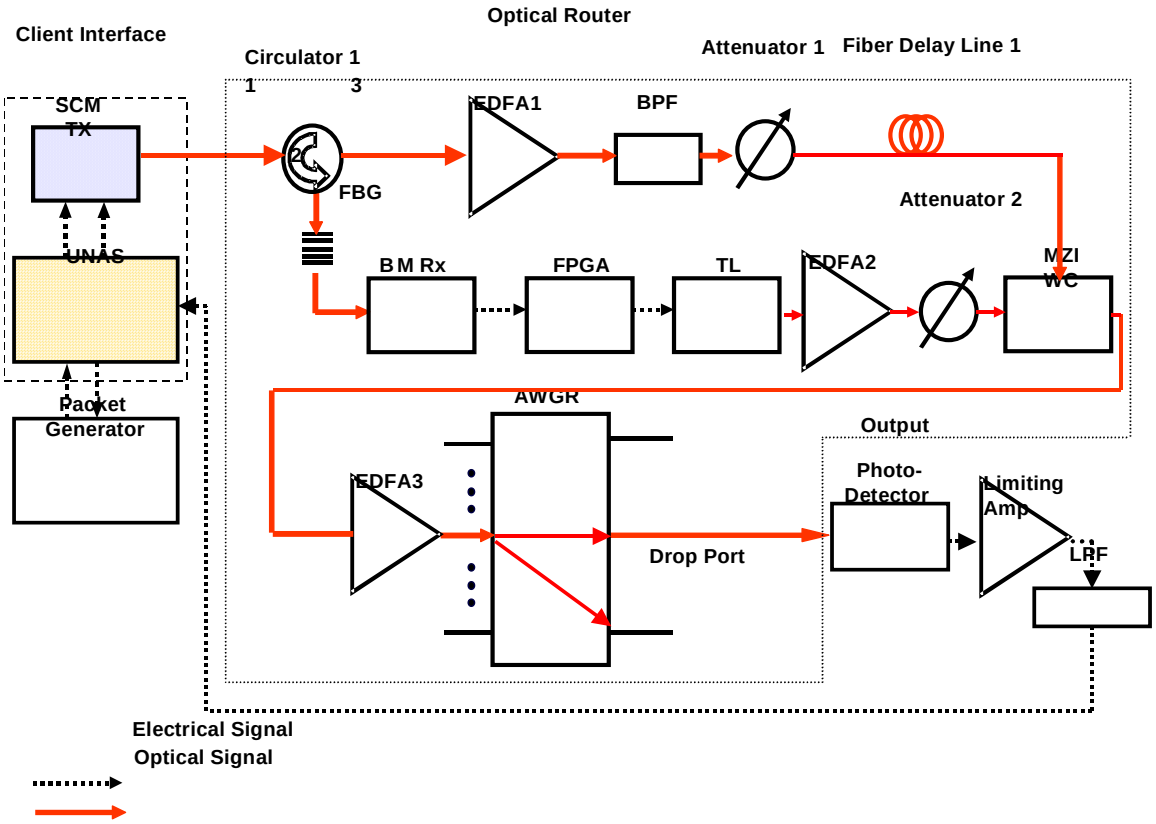
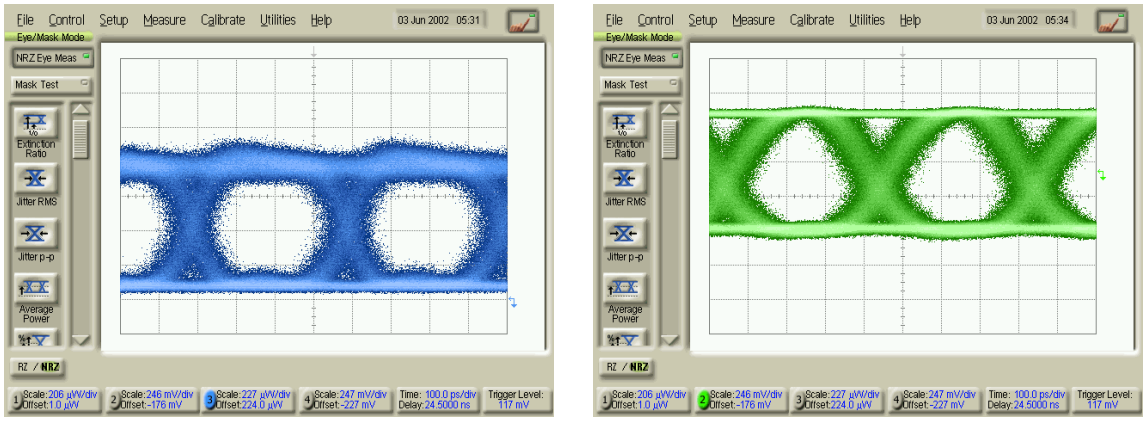


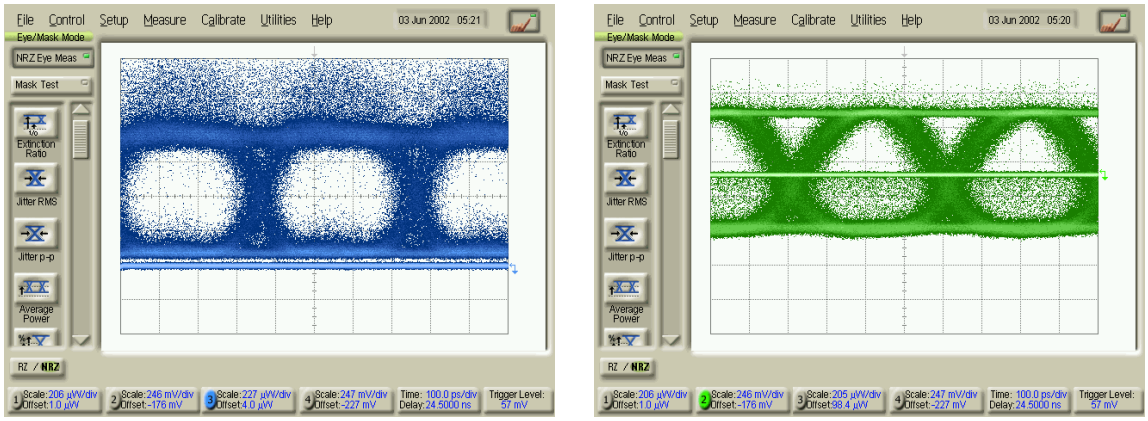
Figure 40 Experiment setup for optical label switching with UNAS, (BPF : band-pass filter, BM Rx; Burst Mode receiver, AWGR; Arrayed waveguide grating router, FBG; Fiber Bragg Grating, TL; Tunable Laser, MZI WC; Mach Zehnder Interferometer Wavelength Converter, L

The eye diagrams of the received signals following the optical router (OLSR) were measured. Figure 41 shows the eye diagram of the output signal when the tunable laser stays at 1552 nm. Figure 41(a) shows the eye diagram of the optical signal and (b) shows the eye diagram of the output signal after being received and amplified by an electrical limiting amplifier. Figure 42 shows the eye diagram of the final output at 1552 nm while the tunable laser is switching between 1546nm and 1552nm. It shows a lot of dots in the eye. The bits at the beginning are stuffing bits, while the bits after a transient period are clear, so the distortion does not introduce packet error (this is not obvious from the eye diagram). Figure 42(a) shows the eye diagram of the optical signal at 1552 nm while the tunable laser is switching between 1546nm and 1552nm, and (b) shows the eye diagram of the output signal after amplified by an AC-coupled electrical limiting amplifier.



(a) **(b)**

Figure 41 Eye diagram of the output signal when the label stays at 1552 nm (a) optical eye diagram (b) electrical eye diagram after amplified by an AC-coupled electrical limiting amplifier. (100 ps/div)



(a) **(b)**

Figure 42 Eye diagram of the final output at 1552 nm switching between 1546nm and 1552nm (a) optical eye diagram, (b) electrical eye diagram after amplified by an AC-coupled electrical limiting amplifier. (100 ps/div) (Dots inside the eyes arise from the distort

The stream of test packets were derived from an IXIA packet generator. The IXIA allowed the logging of packet loss rate. The PLPE FPGA on the UNAS was used to gather packet error rates. Because half the packets are dropped by the OLSR (labels are alternating as $L_1\ L_2\ L_1\ L_2 \ldots$) as described above, half of the total packets should arrive at the IXIA to be counted. When 1546nm packets are sent back, among 1,053,625 packets sent we receive 526,810 without any error. Thus the packet loss rate is 4.75 per million, the packet error rate is 0. When 1552nm packets are sent back, among 1,047,841 packets sent we receive 523,657 with 12 packet errors. Thus the packet loss rate is 0.050%, the packet error rate is 22.9 per million. These rates show that the UNAS-OLSR combination can work together with a very good performance.

This experiment was performed with just a few meters of fiber between the OLSR and the photodetector. Another experiment was run with 50km of fiber inserted between the Mach-Zender Interferometer AWGR wavelength router element. The above experiment was repeated, sourcing packets from the IXIA through the UNAE and OLSR setup shown in Figure 40, but with an additional 50km of single mode fiber insaerted between MZI-WC and EDFA3. Under these conditions, when 1552nm packets are sent back, among 1,518,966 packets sent we receive 758,884 with 84 packet errors. Thus the packet loss rate is 0.079%, the packet error rate is 0.011%. These rates show that the UNAS-OLSR combination with 50 km fiber transmission can work together with a very good performance.

6 Concluding Remarks

Prior to the Universal Network Access System project, much effort had been focused at addressing very high bandwidth network pathways and switching. Less progress had been made on technologies needed to provide transparent, service-independent, high bandwidth network access. The scope of this program is to provide the capability for flexibly interconnecting a wide range of broadband information sources and local area networks over the Next Generation Internet (NGI). The primary objective has been to develop the architecture, components, and a prototype system to demonstrate a Universal Network Access Engine. The engine architecture will enable reconfigurable network monitoring and network edge devices, which can flexibly interconnect broadband information sources and appliances over the NGI. The technology developed provides the groundwork for rapidly deployable and reconfigurable networks, interconnection of dissimilar network types, development and testing of new protocols, packet label or tag generation, network monitoring and quality of service (QoS) testing, and new approaches to data security.

6.1 Architecture

This program undertook to research architectures and methodologies needed to provide a highly flexible approach to realizing network access and payload processing elements. An architecture was developed that would accommodate common elements of protocols in significant commercial use, yet would provide the flexibility to allow development and testing of customized and proprietary protocols.

The strategy was to combine elements supporting bit-rate adaptability and protocol reconfigurability into a system architecture that would allow building a variety of network access adaptors and interfaces. Three functional blocks that were developed were a bit-rate agile DeMux/Mux, a Physical Layer Protocol Engine and a Cell/Packet Engine. The Physical Layer Protocol Engine provides a flexible means of handling physical layer and transport functions including termination of physical links, scrambling/descrambling, bit and Byte alignment and framing and recovery of packetized or formatted data. The Cell/Packet Engine does processing at the packet or cell level, dealing with verifying integrity of data payloads, timing synchronization between user and network clock domains, and processing of user data. This is not only a flexible key to interfacing between user data and the network, but it provides powerful support for determining quality of service. The bit rate agile element was not fully implemented, although the design was thoroughly investigated and substantially completed.

A combination of dedicated integrated circuits and high performance programmable logic were chosen to comprise the engines. Design and development tools were surveyed and selected. The program required balancing the ease of use of tools with higher levels of abstraction and design automation, against the high performance demanded of the engine elements for high bandwidth protocol processing. After a survey of products and approaches, including several interviews and product demonstrations, Protocol Compiler

and VERA from Synopsys were selected for higher level logic design and verification tools, respectively. The Protocol Compiler designs, although more quickly realized, took significant added design time to optimize for performance and to debug, since the abstraction was forced on the designer. In contrast, Verilog took longer for initial design; but the design approach allowed more direct control of performance, showing that for these very high performance designs, the Verilog is superior to Protocol Compiler.

The architecture combining the proposed elements was completed; and a series of card-level Universal Network Access Engines developed. These share common hardware and are reconfigured through software commands and a VxWorks embedded operating system. A set of mezzanine cards was developed to meet the requirement of attaching to various physical media. Each attaches simply to a motherboard to form a UNAE element.

These engines were integrated into a simple, multiple card system architecture, which leads to several advantages: By reconfiguring in approximately one second under software control they provide the ability to rapidly and inexpensively deploy edge elements for a variety of networks using a base set of hardware. They lay the groundwork for the development and testing of new protocols where appropriate. Work in collaboration with the University of California (UC) at Davis to develop a custom gateway protocol to an all optical routed network showed the flexibility of the UNAS architecture. This collaboration included development of a label-filtering protocol, where a custom label was created from IP header information and sent in a reduced bit rate stream to an optical label processor to generate optical subcarrier multiplexed labels for an all-optical packet routing switch.

Engines also provide flexibility in providing scalable network access for a variety of different services and data sources. This was demonstrated by providing interfaces from both standard definition (SD) and high definition (HD) digital video sources and two common types of packet based, high bandwidth transport networks (Gigabit Ethernet and OC-48 Packet over SONET).

6.2 BRAD ASIC

The approach of using a mezzanine board for physical media attachment and the serialization/deserialization and clocking functions was very powerful. A key element of the mezzanine was to be the bit-rate adaptive DeMux/Mux (BRAD), which would provide bit-rate agile transmitter and receiver clock circuits from 150Mbps to 3 Gbps, burst mode capture, and variable parallel word widths on the deserialized side. A CMOS feasibility study based on available processes and models suggested that the risk would be too high in CMOS. A BRAD design based on the IBM 5HP SiGe process was developed and simulated. This design was discontinued due to resource issues. In its place a commercial clock and data recovery and DeMux/Mux chip was successfully designed into the mezzanine. All testing was done with the commercial chip. Without the BRAD, video interface mezzanines also needed custom chips to handle the video signals. This led to successful implementations that required separate optical network-, SD video- and HD video-mezzanines.

6.3 Results

Interfaces and protocol- or format-processing engines were successfully realized and tested for OC-48 Packet over SONET (POS), Gigabit Ethernet, HD video and SD video. That these can all run on the same card with only a few-second SW configuration show the strength of the architecture and approach using programmable engine cards with physical layer mezzanines. In combination, these cards also showed the ability to accommodate generally formatted user data, the ability to quickly select and transmit from a library of protocols, the ability to function at several different data rates (bit-rate adaptability) and the ability to translate between different formats or protocols including SONET to/from Gigabit Ethernet, video mapped to/extracted from SONET or Gb Ethernet. Packet and network diagnostics were also demonstrated, through reporting of SONET alarms and link status indications but also through packet and payload diagnostics within processing elements in the UNAE itself.

The system was used to communicate directly with other attached network elements over a wide area network, talk to routers through PPP/LCP for Packet over SONET and auto negotiate for Gigabit Ethernet. The packet filtering and negotiation for this was handled autonomously on the UNAE card under control of its embedded operating system. The system fully negotiates for link connection with commercial test equipment such as IXIA and Adtech packet testers.

The ability to set up links with other network elements and UNAE multiprotocol mapping were combined to demonstrate for the first time the transport of uncompressed (SMPTE 292) HD video data at 1.5Gbps over a wide area IP packet based network. This was demonstrated on several occasions from Seattle to Denver and to Washington DC over the Internet2 backbone, with diagnostics for dropped and reordered packets obtained continuously for up to 18 hours at a time. Besides showing for the first time that this could be done, this provided unprecedented validation that routers on the network could handle extremely large, single flows of real time traffic such as these.

Several types of link and network diagnostics were performed in real time on the packets as they streamed through the engine cards. Packet ordering errors, dropped packets and some level of packet inter-arrival time jitter were extracted and reported to the user. This is just the first step in addressing quality of service measurements in IP networks with the UNAS approach.

A final example of protocol flexible performance using the UNAS is the collaborative development with University of California at Davis of a gateway interface between an OC-48 POS network and the Client Interface for an Optical Label Switched Router (OLSR). IP packets were extracted from the OC-48 stream, the packet header information was used to generate a label that instructs the OLSR how to switch that packet, and the label and original packet data are forwarded to an optical label generation module. Customization of the stream such as encapsulation using HDLC for data scrambling and appending a preamble are also programmed into the data processing in the UNAE path. The UNAS also receives optical data from the Optical Label Switched Network and recovers the packets and data. This is another unprecedented demonstration, that of a

completely custom and extendable interface between a conventional POS network and an Optical Label Switched Network. The architecture of the UNAE can be extended with the OLSR to other advanced developments such as generation of jumbo packets in concatenated streams.

6.4 Issues

The program did have some setbacks, including the decision not to complete the BRAD within the program. This chip would have been one of the first burst mode receivers to run at 2.5Gbps. This would have eliminated the need for the cumbersome preambles in the OLSR client interface. The BRAD would allow a single mezzanine board to be used for all the protocols and formats addressed in the program, using a multiplexor selector between a fiber and coax (video) inputs.

One of the biggest design challenges was to simultaneously accommodate several clocks (not multiples of each other) such as SONET and Gigabit Ethernet. Having the BRAD would have addressed this most effectively. However, in the implementation without the BRAD, the performance was limited by the components used. Failure for the vendor to successfully build oscillators within spec after our mezzanine redesign limited use of the system to run in “loop-time” (deriving the transmit clock from the received clock) rather than “local-time”, where an on-board oscillator provides it. This did not impact any of the demonstrations of features but limits the applications unless a suitable oscillator is added.

Use of Protocol Compiler tool was considered to have added some time to the full design task, since the performance demanded of the programmable logic by this program required careful redesign of each module within Protocol Compiler to assure consistent performance in the integrated designs.

6.5 Summary

The intent of the research has been successfully carried out. A multirate and multiprotocol system, based on software configurable, hardware accelerated processing engines has provided a series of system demonstrations that show the power and flexibility of the concept, architecture, and system approach. There are still areas where additional work would be of interest:

Successful implementation of each of the protocols in Table 1 was completed with the exception of ATM over SONET and lower rate SONET transport. It was deemed by DARPA representative Mari Maeda a higher priority to implement the two digital video formats and adaptation of those to Packet over SONET, in order to demonstrate high performance multimedia transport on IP networks (something which hadn’t been successfully done at the levels achieved here). Extension of the UNAS engines to accommodate the lower rates would extend the application of the system somewhat, although the cost of doing so may not warrant the effort. SONET OC-12 and -OC3 are in commonplace use at this time.

Completion of the BRAD would be interesting to validate the design and provide a single mezzanine-level front end for the variety of protocols of interest. Its performance as a

burst receiver capable of 2.5Gbps could be of significant value until a commercial chip with such capability is available. However, the BRAD is limited to applications that can tolerate 0.25 IU of jitter, since the basic design only uses a four phase clock.

Work on the IP to Optical Label Switched Router network gateway could be extended, since only the simplest adaptation has been demonstrated. Creation of jumbo IP packets and integration of Network Management and Control functions would be useful in continuing development of OLSR based technology. There are no major issues in realizing these capabilities in the UNAS architecture.

The demonstration of transport and full recovery of very high bit rate (1.5Gbps) digital video data over wide area networks using Packet over SONET demonstrates the concept of real time transport of high bandwidth serial data over IP networks. This could be extended to include some level of error concealment or error recovery. No work was done using techniques such as forward error correction or packet replacement, although packet order error correction has been demonstrated.

7 List of Symbols, Abbreviations, and Acronyms

Abbreviation or Acronyms	Description
AIS	
ARP	Address Resolution Protocol
CAIDA	
CDR	
CPE	Cell Packet Engine
CRC-32 or CRC-16	Cyclic Redundancy Check, 32 bit or 16 bits, respectively
DA	Destination Address
DCM	Digital Clock Manager in Xilinx FPGA
EOP	End Of Packet
FCS	Frame Check Sequence
FIFO	First In First Out memory
FPGA	Field Programmable Gate Array
GbE	Gigabit Ethernet
GbEI	Gigabit Ethernet Interface - Refers to the GbE implementation on UNAE Hardware
GMII	Gigabit Media Independent Interface
HDLC	High Data Layer
IETF	Internet Engineering Task Force
IPG	Inter-Packet Gap
LSB	Least significant bit
MAC	Media Access Control

MSB	Most significant bit
NGI	Next Generation Internet
NTON II	National Transparent Optical Network testbed
OIF	Optical Internetworking Forum
PCS	Physical Coding Sublayer
PHY	Physical Interface
PMA	Physical Medium Attachment
PMD	Physical Medium Dependant
POS	Packet Over SONET
POS-PHY	Parallel physical layer bus proposed by SATURN users group to accommodate POS data on the user side. POS-PHY level3 continuously handles data payloads at OC-48 data rates.
RTP	Real Time Protocol – part of UDP/IP stack
SA	Source Address
SERDES	Serialization and deserialization function. Fully equivalent to mux/demux mux/demux – Multiplex and demultiplex function.
SFD	Start Frame Delimiter
SMPTE	Society of Motion Picture and Television Engineers
SOP	Start Of Packet
SPE	Synchronous Payload Envelope (typically SONET)
UDP	User Datagram Protocol – part of UDP/IP stack
UNAE	Universal Network Access Engine
UNAS	Universal Network Access System

8 Index

9 References

[i] Compact PCI PICMG 2.0 Revision 2.1 dated September 2, 1997

[ii] Turner, Jonathan S. “Terabit Burst Switching” Journal of High Speed Networks, 1999.

[iii] S. J. B. Yoo, Hyuek Jae Lee, Yanda Zhang, S. K. H. Fong, Vincent Hernandez, Vincent K. Tsui, Srikanth Vaidianathan, Katsunari Okamoto, Shin Kamei, "High-Speed All-Optical Packet Routing System with Optical-Label Switching and Optical-Label Swapping using Rapidly Tunable Wavelength Conversion and a Uniform-Loss Cyclic Frequency AWGR," to be published in IEEE photonics technology letters, August 2002.

[iv] Mihai Banu, Alfred E. Dunlop, Wilhelm C. Fisher, and Y. Ota. “Design Techniquies for Non-Oversampled, Instantaneously Locked Clock and Data Recovery Circuits – An Overview”, IEEE Catalog # 95TH8031.

[v] Optical Internetworking Forum Implementation agreement OIF-SPI3-01.0, System Packet Interface Level 3 (SPI-3): OC-48 System Interface for Physical and Link Layer Devices. This is fully equivalent to the Saturn POS-PHY Level 3 packet interface.

[vi] Draft-ietf-avt-smpte292-video-05.txt, RTP Payload Format for SMPTE292M Video, May 30, 2002.