The Design of an IVDS World Wide Web Browser Architecture

by

Aaron George Hawes

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

Nathaniel J. Davis, IV, Chairman

Scott F. Midkiff

Willard W. Farley, Jr.

December, 1997

Blacksburg, Virginia

Keywords: Interactive Video Data Service, World Wide Web, Browser, FPGA

The Design of an IVDS World Wide Web Browser Architecture

Aaron George Hawes

(ABSTRACT)

An IVDS (Interactive Video Data Service) uses an interactive television system to transmit data to and from subscribers' homes. IVDS allows the viewer to interact with content provided on the television using a remote control. A typical IVDS application would be ordering an advertised product or playing along with a quiz show.

The Virginia Tech Center for Wireless Telecommunications (CWT), under a contract with Interactive Return Service, Inc., is developing an IVDS system in which content is provided through the television cable system in the form of audio codes. A special remote control can detected these audio codes and query the user for input. The return path for this system is a wireless channel. The remote control contains a spread spectrum transmitter that transmits packets to a Repeater unit residing within a quarter mile of the user's home.

With the popularity of the World Wide Web soaring, many companies are announcing internet appliances that will bring the content of the web to the user at a fraction of the cost of a standard personal computer. CWT has been contracted to extend the core IVDS system to provide a web browsing capability, allowing the user to browse the web with only the remote control.

This thesis outlines the requirements of the IVDS Web Browser System. The different hardware design concepts are documented. The final Browser System specification is presented, as well as a board-level description of the Decoder Unit that is part of this final Browser System. Finally, a detailed description, current status, and simulation results are presented for the FPGA (Field Programmable Gate Array) that serves as the controller for the Decoder Unit.

Acknowledgments

I would like to thank my Dr. Nathaniel Davis, my advisor, for his guidance throughout this research and the writing of this thesis. I would to thank the Center for Wireless Telecommunications for allowing me to participate in this IVDS Project. I would like to thank Woody Farley and Dr. Scott Midkiff, members of my committee, for their support and guidance throughout this research.

I would like to thank my fiancé, Trish, for her love, support, and patience throughout my research and writing. I would like to thank Dr. Boris Davidson for all of his advice on issues surrounding this project. I would like to thank my parents Louis and Georgia Hawes, and my brother Todd, for their encouragement.

Finally, I would like thank my three ferrets, Killian, Saranac, and Celis, for their participation in the writing of this thesis by constantly walking and lying on the keyboard, encouraging the frequent use of the *Delete* and *BackSpace* keys.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction to the Internet Appliance

With the spread of internet use across the country, there is a new push to bring the wealth of information on the internet to an untapped market of users uncomfortable with the price and technical issues associated with a personal computer. Numerous companies involved in the hardware and software computer markets in this country have announced products targeted toward this new internet appliance market. Futhermore, many television manufacturers have announced the planned incorporation of these devices inside television sets to provide simplified internet access.

Companies from Netscape (through spin off company Navio) to Oracle to Sun Microsystems to Microsoft (through the acquisition of WebTV) to Apple to Sega to Gateway are introducing low cost internet devices that provide a minimal hardware set with the goal of providing easy to use, cost effective platforms whose sole purpose is to access the internet. The estimated price point for consumer oriented versions of the internet devices lies somewhere between $300 [1] and $500 [2]. With this broad spectrum of industries involved it is no longer a matter of if, but when internet devices become part of the average household.

1.2 The Interactive Video Data Service Project

While most of these planned internet devices require a telephone line and an internet service provider, a small startup company, Interactive Return Service (I.R.S.), has begun to develop a product intended to bring both interactive television and the World

Wide Web into the home at a very low price, without requiring the use of the user's telephone line.

The Interactive Video Data Service (IVDS) project began roughly two years ago when I.R.S. C.E.O. Fernando Morales came to Virginia Tech's Center for Wireless Telecommunications with the idea for a wireless consumer product. This project, involving numerous professors and graduate students, has now reached an early prototype status with many new promising applications for this system envisioned.

This basic system concept begins with an audio segment of television. Within the audio track of the segment, IVDS audio code is embedded. This audio code is placed in the programming segment at a location that is very difficult for the user to hear but a special IVDS remote control, the Audio Link, can recognize.

This audio code may contain information about the programming segment or control information for the Audio Link. The Audio Link contains an audio speaker that can be used to ask a user to respond by pressing certain buttons on the Audio Link. Content can be placed in the audio code so that the user can interact with the programming content he or she is currently watching.

An application of this system involves a commercial containing an audio code that relates to the advertised product. The audio code contains some form of identification for this product as well as information as to how the user may respond.

The Audio Link would detect the audio code and respond by asking the user to press a particular button for more information regarding this product or in a home shopping scenario, to order the product.

The Audio Link would generate a packet containing information about this product and a code identifying the user. The Audio Link contains a wireless spread spectrum transmitter that would transmit the packet over a wireless, simplex channel in the 900 MHz unlicensed frequency band.

An IVDS Repeater (also designed and in the prototype phase as part of the IVDS Project) residing in the neighborhood within roughly a quarter of a mile of the Audio Link trsnamitter would receive this transmission. The Repeater would forward the contents of this packet across a public network via a modem connection to a large host computer that would process the user's message appropriately.

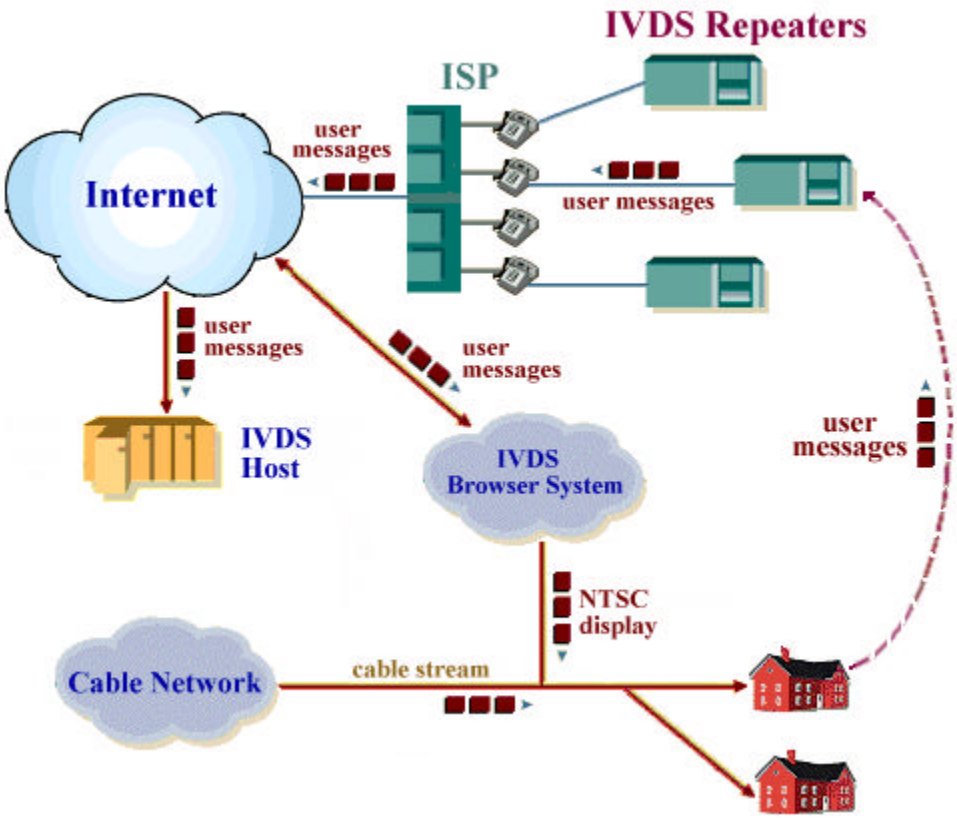


Figure 1.1: IVDS System with IVDS Browser System Extension

As an extension of the core IVDS project, the Center for Wireless Telecommunications has also been contracted to design an additional system that would use the IVDS System and provide the ability to browse the internet requiring only the television and the Audio Link. This IVDS Browser System, shown in Figure 1.1, will accept user input from the core IVDS system, retrieve the web pages from the internet, and present the web page audio and visual display on the user's television. This display will require the allocation of a particular channel for this purpose by the local cable television provider.

A typical scenario would involve a user wanting to browse the Weather Channel's web site to find out what the weather will be for that day. The user would tune to the "web channel" on his or her television (or cable box) and press the button on the Audio Link to initiate a web browsing session. The Audio Link would generate and transmit a packet containing the message to initiate this web browsing session. The Repeater would recognize the received message as a web browsing session request and forward the user's command to the Browser System. The Browser System would process the command, respond accordingly, and display the results of the user's command on the "web channel" of the user's television.

1.3 Thesis Organization

This thesis will introduce the requirements of this IVDS web browsing system and detail the evolution of the IVDS World Wide Web Browser System concepts into a final system design. A board-level description of the Decoder, as part of this final distributed

IVDS Web Browsing System architecture, will be presented. A detailed description of the logic design within the FPGA (Field Programmable Gate Array), which controls the Decoder operation, will be provided. Finally, this thesis will detail the status of this design and the remaining work to be completed in order to implement the IVDS Web Browser System.

Chapter 2 will give a general overview of the visual, audio, network, CPU, and operating system requirements of the Browser System. Chapter 3 will introduce the four preliminary Browser System design concepts, discuss the issues that needed to addressed in order to finalize each design, and provide a brief explanation why each design was not used. Chapter 4 will introduce the final Browser System design. Chapter 5 will discuss the Decoder of the final design from the board level. Chapter 6 will detail the design of the FPGA for the Decoder. Chapter 7 will present the problems encountered with the Decoder design and the resulting design modifications. Chapter 8 will discuss the current status of the FPGA design and provide a discussion of the timing diagrams found in Appendix C which demonstrate the current functionality of the FPGA design. Chapter 9 will provide a conclusion to this thesis.

Chapter 2. General Browser System Requirements

2.1 Introduction

This chapter will provide a general description of the functionality of the IVDS WWW Browser System and some of its hardware requirements. I will not go into great detail on the system requirements because they are greatly influenced by each Design Concept. The next chapter will discuss each Design Concept in detail, the issues involved, and its relationship to the overall system requirements.

2.2 Browser System Network Interfaces

2.2.1 User Input to the Browser System

The IVDS Repeaters will recognize certain messages from the user as web session messages and will forward these "web messages" to the Browser System. The Browser System will need a network interface to one or more Repeater units. It will not be necessary for the Browser System to know from which Repeater a particular web message originated.

A web message will contain some form of user identification, as well as an identification of the button on the Audio Link that has been pressed. Each button on the Audio Link will correspond to a particular browser function such as following a link, scrolling, or tabbing between links. The software issues involved in processing user commands are outside the scope of this thesis.

A single Repeater will potentially serve from ten to hundreds of homes. Therefore, it is clear that a one-to-one correspondence between Repeaters and Browser Systems is not possible. The Browser System must be able to accept input from multiple Repeaters.

2.2.2 Browser System Internet Access

The Browser System will need access to the internet in order to retrieve the requested web pages. For each Design Concept, the Browser must be able to support TCP/IP to establish connections with web servers on the internet.

2.2.3 Modulating the NTSC Display into the Cable Stream

The cable stream network interface will allow the composite audio and video NTSC signal to be multiplexed into the cable television stream within the 6 MHz bandwidth of a single channel. The cable television stream will flow into the user's home where the television tuned to the "web channel" will be able to view the web page and hear the accompanying audio, if present. The user will then be able to interact with the web page using the Audio Link.

The components required to interface to the cable stream would be common to each of the Design Concepts. Because of the commercial availability of off-the-shelf parts providing this functionality, this interface was always the last consideration in the Design Concepts. As a result many of the Design Concept descriptions will not mention this interface because the design was discarded in the specification stage. The specification of

these hardware components will not be presented until the final Design Concept in Chapter 4.

2.3 Visual Display Requirements

To display information on a television, the Browser System will first render the retrieved web page into the RGB (Red Green Blue) pixel format at a resolution of 640x480 pixels. An NTSC (National Television Standards Committee) encoder will be used to convert the pixels into the NTSC video format.

The display presented by the Browser System must allow a user to be able to sit across the room and comfortably read the text of web pages on the television. The resolution of the displayed text will need to be large enough for a user to comfortably read from the television for an extended period of time. Displaying computer graphics on a television at a resolution higher than 640x480 results in various forms of distortion that can make text difficult to read. All of the NTSC encoders reviewed for this project are designed to display the 640x480 pixel resolution.

2.4 Audio Requirements

The Browser will also be designed to forward a certain amount of sound with embedded audio codes to the user. It is desirable that the Browser be able to send several seconds of audio to the user. This will enable the audio code to be embedded within the audio segment so as to make the audio code inaudible to the user.

The Browser System will need to send an audio code to the Audio Link when the user first initiates a web session. Upon receipt of this audio code the Audio Link will transmit packets with a higher priority bit set to the Repeater so that the Repeater will forward them to the Browser System with minimal delay.

Furthermore, when the user finishes the web session, an audio code must be sent to notify the Audio Link that the web session has ended. Thereafter, the Audio Link will transmit normal priority packets until otherwise instructed by an audio code.

2.5 Conclusion

This chapter presented a general description of the network interface requirement as well as the visual display and audio requirements of the Browser System. This level of detail was the point at which the research of the various Design Concepts began. While various functionality was added and removed across the Design Concepts, these general requirements remained the constant.

Chapter 3. Preliminary Design Concepts for the IVDS Browser System

3.1 Introduction

This chapter will present each of the four IVDS Browser System Design Concepts that preceded the final design. A system description will be provided with each Design Concept as well as a discussion of the hardware and associated issues involving the design. Finally, the reason will be presented for each particular design being discarded in favor of the next design.

3.2 Design Concept 1

3.2.1 System Description

When the project began, the IVDS Browser concept was a simple, stripped down personal computer with a single 386 or 486 processor, no hard disk, a small amount of RAM, a cable modem, an NTSC encoder, an audio processing hardware unit, and a composite video generator to modulate the combined audio and video signal into the cable television stream.

The prototype hardware would be initially pieced together using off the shelf components and tested. The final design was to be consolidated into a system encompassing a minimal number of custom boards.

At the system level, it was assumed that the IVDS Repeaters would have some kind of network connection to send user commands to a cable system headend where a cable stream manager would forward the commands through the cable system to the appropriate IVDS Browser. The equipment and coordination at the cable system headend

required to do this would be dependent upon the individual cable provider's system architecture. The sponsor directed us to assume that the user commands would be passed through the cable system and received by the intended Browser using a cable modem.

Retrieval of web pages would require a bi-directional path between the Browser and the headend. The physical paths through which the cable modem achieved duplex communications with the headend would depend upon the cable provider's cable system architecture. Currently, many cable providers are in the process of upgrading their residential networks from the traditional all-coaxial cable simplex network to a hybrid fiber/coax duplex network. Commercially available cable modems at the time of this research provided the availability to function on either on these network topologies.

The down stream path for both topologies is the cable stream as shown in Figure 3.1. The upstream path for the fiber/coax duplex networks is also the cable stream itself. However, for traditional cable topologies there is no upstream path within the cable stream. To solve this limitation, some cable modems provide a feature called the telephony return option that allows the use of a dedicated telephone line as the upstream path. This feature will allow the IVDS Browser to be used with either cable network topology.

The Browser would receive user commands through the cable system, process the command, and transmit the web page request upstream using either the cable stream or the telephony return option. Equipment at the headend would receive the request from the upstream path and an IP router would forward the request out onto the internet over a

dedicated line. Requests made via the telephony return option would be forwarded to the equipment at the cable system headend and would be sent out onto the internet as well.

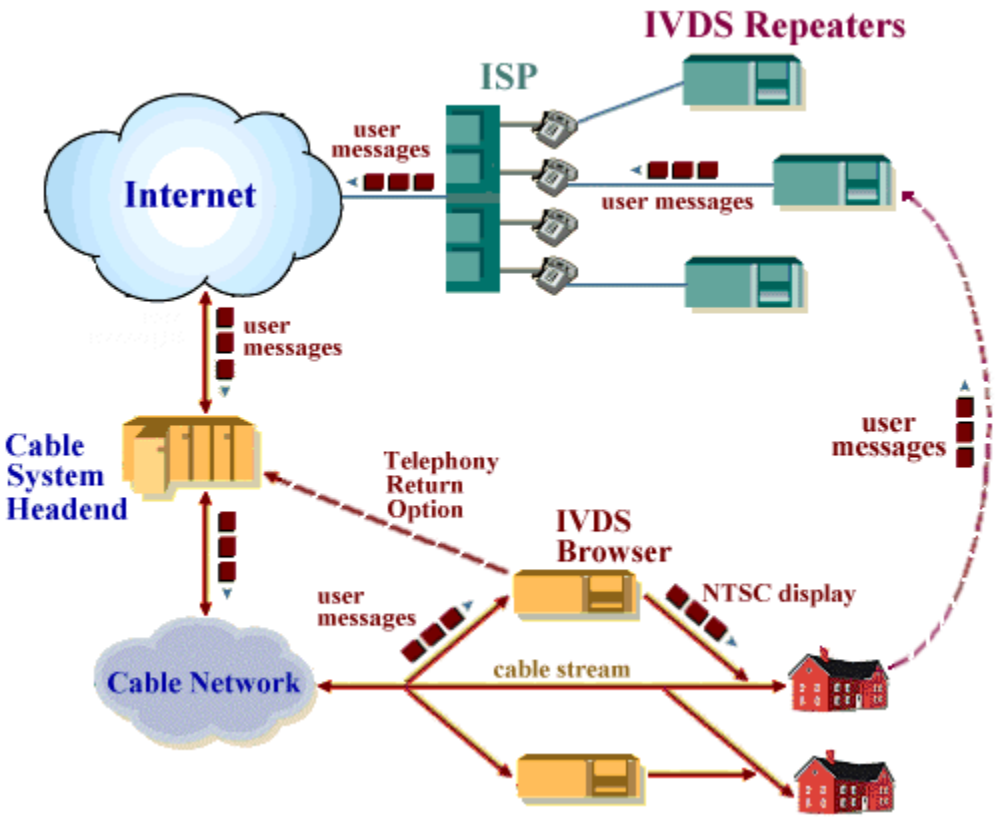


Figure 3.1: Design Concept 1 System Diagram

The return data would be forwarded by the router at the headend down the cable stream to the Browser. The transport layer connection could be maintained to a remote internet web server using either of these return paths to the cable system headend. Transmission of data over a hybrid fiber-coax network will conform to the IEEE 802.14 specification [3].

Web pages received from the internet and visual updates to the web page display itself (such as displaying a focus change from one hyper link to another) would be

rendered graphically using a VGA chip set at a 640x480 resolution. The VGA output would be directed to an NTSC encoder that would convert the RGB pixel format into the NTSC format.

Once converted into the NTSC format, data would be input to a composite video generator. The composite video generator would accept both the NTSC and analog audio signals, combine them, and modulate the resulting signal into the cable within the bandwidth spectrum allocated to the IVDS "web channel." Once in the cable stream, any television down stream from the Browser that was tuned to the "web channel" would be able to see and hear this display.

It was envisioned that a single Browser unit would serve multiple homes using one television channel. If someone was already using the Browser System and another user wished to use the same Browser System, the new user would be placed in a waiting queue.

A user in the queue would be able to use the system when other users ahead of them in the queue had finished their browsing sessions. The implementation of this queuing functionality would be a software issue within the Browser itself.

3.2.2 Operating System

The CPU within the Browser would need to support the processing of user requests, web pages, graphical rendering, and the TCP/IP network protocol for internet access. This set of functionality would require a multitasking environment, necessitating the inclusion of an operating system to manage these different tasks. Because the

decision had already been made that no hard disk would be present in the Browser, the operating system would reside in a ROM.

The required operating system would need to be a multitasking environment with support provided for graphics and a TCP/IP stack. If the decision was made to support Java, the operating system would also need to be multithreading. Additionally, the operating system would need to have a small kernel size, minimal processor and memory requirements, a modular design so that functional pieces could be added and removed as desired, and an inexpensive licensing cost.

Our research group had the option of pursuing an off-the-shelf operating system that could support our needs or we could write our own operating system. Writing an operating system for this project provided the benefit of the ability to control the source code and optimize it for our particular application. However, the long development time of an operating system as well as maintenance costs if the hardware platform should change, might require more resources than could be allocated to the project.

Licensing an operating system would allow the project to leverage an already established operating system. The advantage of licensing an operating system would be the ability to insert and remove software components such as networking and graphics libraries from third party vendors and allow much shorter development time.

The advantages and disadvantages of each option were presented to the sponsor and the decision was made to specify an operation system to license for development once a better understanding of the hardware options was realized.

Several operating systems were reviewed as potential candidates. John Stanhope was the technical reviewer of the operating systems. The table below summarizes highlights of the operating systems reviewed.

Table 3.1: Candidate Operating Systems

Operating System	Mulitasking	Multithreading	Network Support	Graphics Support	Platforms
Embedded DOS	yes	no	yes	yes	x86, Pentium
QNX	yes	yes	yes	yes	386, 486, Pentium
pSOS	yes	yes	yes	yes	PowerPC, 68K
Linux	yes	yes	yes	yes	x86, Pentium, PowerPC, etc.

pSOS met our requirements and had a wealth of third party developer support products. The quotes we received for per unit licensing costs both in low and high quantity were more than for QNX. Since QNX also met our requirements, the sponsor was more interested in pursuing QNX licensing.

3.2.3 Hardware Components

The chosen operating system would dictate the use of a certain subset of microprocessors and graphical and networking chip sets. The processing power requirement would probably necessitate at least a 386 processor. Because of the environmental conditions and other factors, the sponsor did not want to include a hard disk in this system.

The unit to convert the VGA based pixels into NTSC could take one of two forms, an internal NTSC encoder or an external NTSC scan converter. An internal NTSC encoder card would reside inside the Browser itself. This internal NTSC encoder card would probably also contain the VGA chip set to render the web page.

An external NTSC scan converter is a small box that would be attached to the output display port of the VGA card. The output of the NTSC converter would be fed into an RF modulator to be multiplexed into the cable stream.

The cable modem interface to the Browser would be dependent upon the cable modem selected. Ethernet 10BaseT is emerging as the most predominant interface method [4].

At the prototype level, this interface would require a standard Ethernet 10BaseT network interface card whose chip set was supported by the chosen operating system. For the final design, a supported Ethernet chip set would need to be specified.

To add audio to the system, we would initially use a sound card to process audio formats found on web pages and eventually specify a chip set to include in the final design. Both the sound card and the final chip set would have to be supported by the specified operating system.

At this junction in the research, an operating system was to be specified and then the hardware components that this operating system supported would be reviewed and specified. The sponsor had developed an interest in the functional components within a DBS (Direct Broadcast Satellite) set-top decoder box. He directed our effort toward looking at the hardware components in this set-top decoder box and how the low price

point was achieved. He was interested in the feasibility of adding the ability to decode MPEG (Moving Picture Experts Group) coded audio and video segments to the Browser System.

Additionally, computer industry publications began to focus on the developing internet appliance systems. An article in Byte magazine prompted the investigation of several new development systems for these network computers and set-top box systems [2].

3.2.4 Design Concept 1 Conclusion

The applications of the MPEG transmission and decoding ability of the DBS set-top decoders and the release of new development packages targeted at the developing network appliance market lead to the arrival at a new Design Concept for the IVDS World Wide Web Browser. The current Design Concept was no longer pursued in this form; the research effort was now directed toward the second IVDS Browser System Design Concept.

3.3 Design Concept 2

3.3.1 Background

This overall system design would remain the same as in the previous design. However, the Browser would be changed from the stripped-down personal computer to a small unit that would more closely resemble the set-top box and network computer designs. The Browser would be able to both browse the web and decode MPEG coded audio and video.

3.3.2 The DBS System and MPEG Coding

This Design Concept evolved from the sponsor's interest in DBS set-top decoder boxes and industry announcement of several set-top box and network appliance development systems. Audio and video are transmitted through the DBS system in the MPEG coding format. At the time this research was conducted, DBS providers had been using MPEG-1 and most had announced that they would be upgrading to MPEG-2. MPEG-1 provides the ability to code video and a single channel of audio together at VHS quality with a bit rate aproximately 1 to 1.5 Mbit/s [5].

MPEG-2 adds the flexibility to support different levels of video resolution at bit rates up to 8Mbit/s. Additionally, MPEG-2 extends the MPEG-1 single audio channel design to support multiple audio channels while remaining backward compatible with MPEG-1. A typical application using multiple audio channels would be surround sound audio [6].

Encoding audio and video into MPEG requires much more processing power than the decoding from MPEG. Furthermore, MPEG encoding chip sets are very expensive. In a DBS system, fewer MPEG encoders are needed because encoding is performed only at the satellite uplink point. DBS systems use a set-top box that contains a MPEG decode-only chip set.

At the time of this research, these set-top boxes performed the decoding from MPEG and encoding into NTSC. Consequently, these set-top box units contained only small processors to manage the different functions.

The sponsor wanted to determine the feasibility of designing the Browser System to decode MPEG coded audio and video in addition to functioning as a web browser. The ability to decode MPEG streams and access the internet were key features promoted by several companies in press releases for their internet appliance development systems.

3.3.3 Acorn 7500 Development System

Acorn Computers Limited had recently announced the availability of its Acorn 7500 Development System [7]. With the addition of Acorn's add-on networking card with TCP/IP support [8], MPEG decoder card, NTSC encoder card [9], the resulting development system appeared to be one potential solution for building the Browser with the MPEG decoding ability. Oracle is the major licensee of the NC technology being developed by Acorn [10].

3.3.4 LSI Logic s INTEGRA™

LSI Logic had announced its INTEGRA™ architecture for designing set-top box like applications and the INTEGRA 1000 Set Top Development Platform (SDP), its first generation of INTEGRA products [11] [12]. This SDP was comprised of three main chips, a network interface module, a MiniRISC CPU with MPEG-transport descrambling chip, and a MPEG-2 decoder chip as seen in Figure 3.2.

The network interface module could be optionally a cable modem module or a DBS modem module. The cable modem chip integrated Quadrature Amplitude

Modulation (QAM) with forward error correction (FEC). The development system was $29,000 and pricing for this chip set was to be $70-$75 in high volume [11].

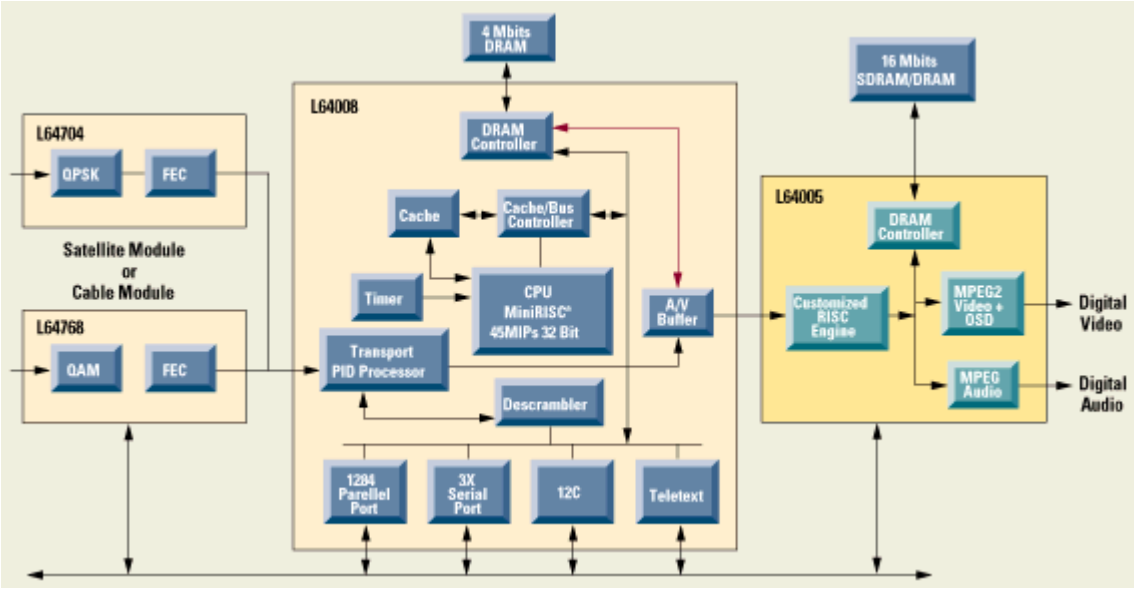


Figure 3.2: Internal Architecture of the INTEGRA SDP [13]

These three main chips were designed using LSI Logic's CoreWare® design program wherein high-level building blocks, or cores, could be integrated together into "system-on-a-chip" designs. Some available cores included Reed-Solomon and Viterbi error correction decoders, single-chip MPEG-2 audio and video decoders, V.34 modems, cable modems, satellite modems, ATM SAR modems, Ethernet, ISDN, VGA, and NTSC/PAL encoder cores [11].

The INTEGRA SDP was a Pentium-based system with the special chip set on a PCI card. Additionally, the SDP was offered with support from third party software vendors in the form of real-time operating systems such as pSOS and DAVID, programming interfaces, and application programs. It was also stated that third party applications, such as "TV-based Internet Web browsers" would be available for use with

the INTEGRA SDP [13]. The inclusion of this statement lead to our investigation to determine whether the INTEGRA SDP could be used for the IVDS Web Browser.

3.3.5 System Description

The sponsor was interested in the LSI Logic INTEGRA SDP solution and directed our effort to investigate the INTEGRA SDP's ability to meet the Browser's needs, as well as inquire into LSI Logic's willingness to design a custom system if the INTEGRA SDP could not meet our needs.

In this second Design Concept, the INTEGRA would be the Browser itself, receiving and processing user commands, fetching and rendering web pages, decoding MPEG audio and video, encoding the rendered web pages and decoded MPEG video into NTSC, and converting the web based and decoded audio into analog audio. The overall system Design Concept remains the same as in Design Concept 1, as shown in Figure 3.1.

Web session user commands would be forwarded to the cable headend as described in Design Concept 1. User commands would be sent downstream through the cable system to the INTEGRA-based Browser. The Browser would receive the user commands through a cable modem network interface module. Commands would be processed and web pages rendered by the RISC based CPU.

It was stated in the INTEGRA Fact Sheet that the MiniRISC microprocessor core was designed to have extra processor cycles for future software upgrades such as graphics pre-processing. MPEG coded audio and video would be decoded by the dedicated MPEG-

2 decoder chip. Video would be encoded into NTSC by the Video Encoder shown in the system block diagram. Audio would be converted into analog by the Audio DAC [13].

3.3.6 Operating System

Several third-party software vendors had created software kits including operating systems for this development system. Integrated Systems Inc. (ISI) would provide a complete software kit for the INTEGRA SDP based on its pSOS operating system. MicroWare Systems would offer its OS9000 operating system and DAVID development. kit [13]. Because we were more familiar with the pSOS operating system, the sponsor directed us to investigate the pSOS offering.

3.3.7 System Hardware Issues

To determine if the INTEGRA SDP was suitable for our Design Concept, several issues needed to be resolved. First, it was unclear how the Browser's functionality described above would be divided among the three chips.

It was stated in the product information that "the powerful 45 MIPs CPU is capable is processing all set-top application needs and control needs. These include ... communication ports protocol processing, remote control processing,..." [14]. This statement would seem to imply that the MiniRisc CPU could process the user commands and support the TCP/IP as well as HTTP protocols.

Figure 3.3 shows a block diagram of the internal architecture of the CPU. The MPEG transport descrambling function simply isolates one particular channel of MPEG

stream data from a larger group of MPEG channels. The isolated channel stream is forwarded to the MPEG decoder chip.

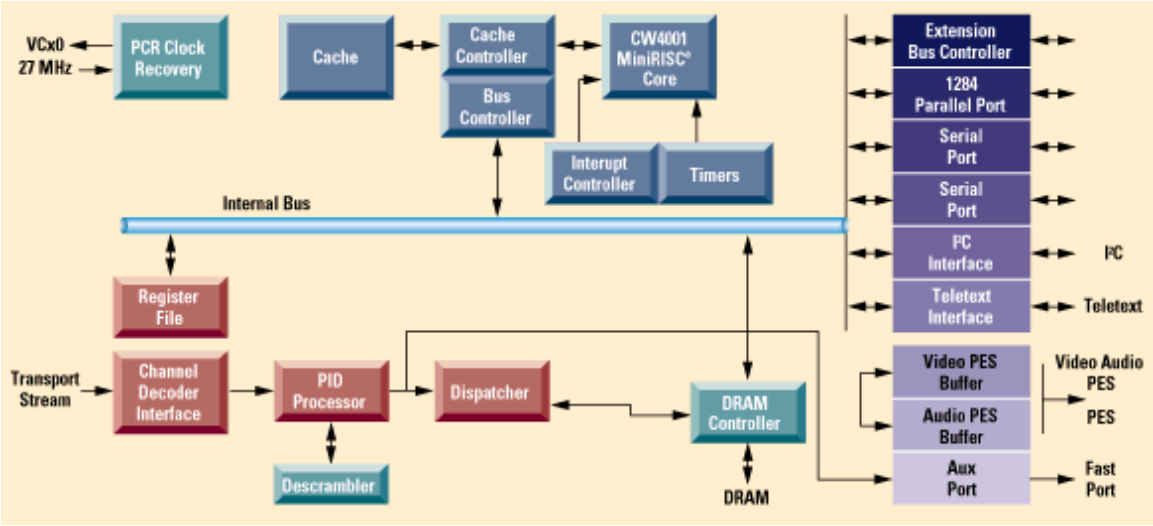


Figure 3.3: Internal Architecture of the CPU and MPEG Transport Descrambler [15]

Second, it was unclear where the graphical rendering would take place and how it would be forwarded to the NTSC encoder. The Features section of MPEG decoder chip's product information contained a section called Video Display and Graphics Controller. Within that section was listed features for the "On Screen Display for program guides and other graphical information" [16].

It seemed reasonable to assume graphical rendering would be performed by this chip. However, it would need to be determined whether this chip could render an entire 640x480 display of a web page. This chip's video output was sent to the video encoder. No internal architecture block diagram could be found for this chip.

Third, it remained unclear whether the cable modem network interface module was a true cable modem with a port that could be plugged directly into the cable stream or if it was a module that interfaced to external cable modem. In Figure 3.4 below, the cable

modem receiver's internal architecture is shown. It was stated in the product information for the cable receiver that the chip was designed to interface to a "standard off-the-shelf analog front end." Additionally, the Features and Benefits section included the statement: "Connects to low-cost tuners" [17].

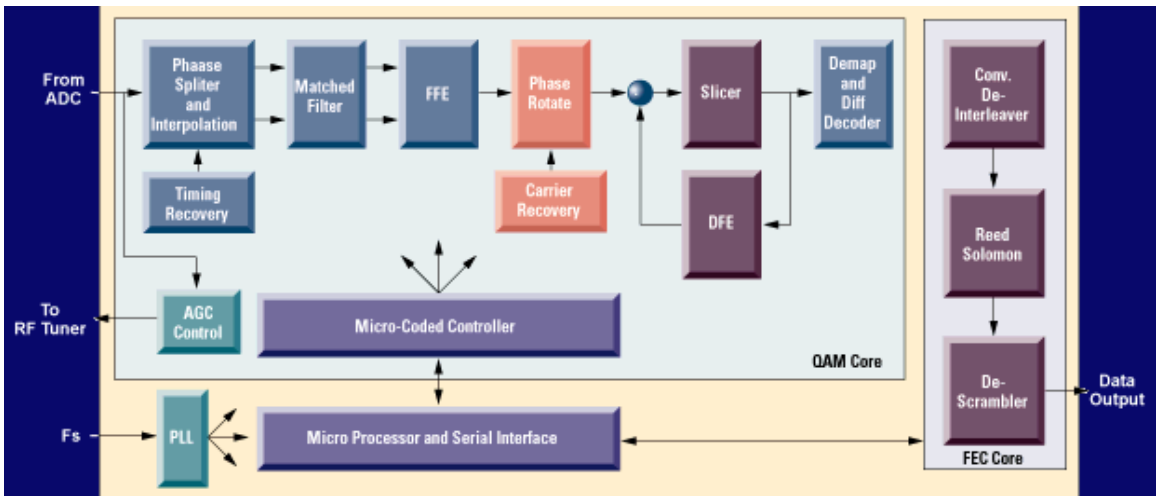


Figure 3.4: Internal Architecture of the Cable Modem Network Interface Module [18]

The product information referred to the cable modem module as a "Single-Chip Cable Receiver" [17]. If third-party software developers were to develop applications such as "TV-based Internet Web browsers," duplex communications with the internet would need to be achieved using some form of return path. It was indeterminate whether this cable modem network interface module was capable of duplex communication or if any form of telephone line based return path was supported.

3.3.8 Design Concept 2 Conclusion

All of the issues above would need to be resolved before any decision on this product could be made. LSI Logic was unresponsive to our attempts to make contact.

Most of the information that we obtained was from the internet and a licensed reseller of the INTEGRA SDP. However, the reseller had limited technical resources to answer my questions.

While the press releases and product information obtained from LSI Logic appeared very promising, the INTEGRA SDP itself did not appear ready for shipment. LSI Logic's "system-on-a-chip" designs were receiving much attention and their custom chip designers were overwhelmed with requests. LSI Logic appeared only willing to entertain requests from companies that were willing to engage in multi-million unit agreements.

The sponsor felt that the LSI SDP availability and custom design facilities could not meet our time table. At that point, the sponsor directed the project toward developing the Browser System with the MPEG decoding and web browsing functions using off-the-shelf parts.

3.4 Design Concept 3

3.4.1 System Description

The overall system design would remain the same as in the previous Design Concepts, as seen in Figure 3.1. The Browser design would be functionally the same as LSI Logic design, however, the Browser would be built using off-the-shelf parts. A cable modem would be used to interface to the cable stream rather than the LSI Logic cable receiver chip in Design Concept 2. The interface to the cable modem would allow reception of messages from Repeaters while allowing connections to the internet.

Connections from the Browser to the cable system headend would be established via either the downstream and upstream cable paths or the downstream cable path and the telephone line return path to the headend.

The Browser would modulate the audio and video web display data converted to an NTSC signal into the cable stream using an RF modulator as described in the previous Design Concepts.

3.4.2 Hardware Issues

The sponsor wanted to design the IVDS Browser with the ability to decode MPEG coded video and audio, as well as browse the internet. In addition, he was interested in accelerating the development process by dividing the Browser System into functional blocks which could be developed separately, but provide the desired functionality when put together.

This led to two internal hardware designs that were similar to the first browser Design Concept but added an MPEG decoding chip as seen in Figure 3.5 and Figure 3.6 below. In the first design, the cable modem interface block would pass data onto the CPU which would be responsible for processing user commands, maintaining the TCP/IP stack, fetching web pages from the internet, processing and forwarding audio to the DAC when needed, and optionally passing MPEG streams onto the MPEG decoder.

Unlike the INTEGRA SDP MPEG decoder chip, the MPEG decoder's only function in this system would be to decode MPEG audio and video streams. This meant that an off-the-shelf MPEG hardware decoder could be used. A block diagram of this scenario can be seen in Figure 3.5.

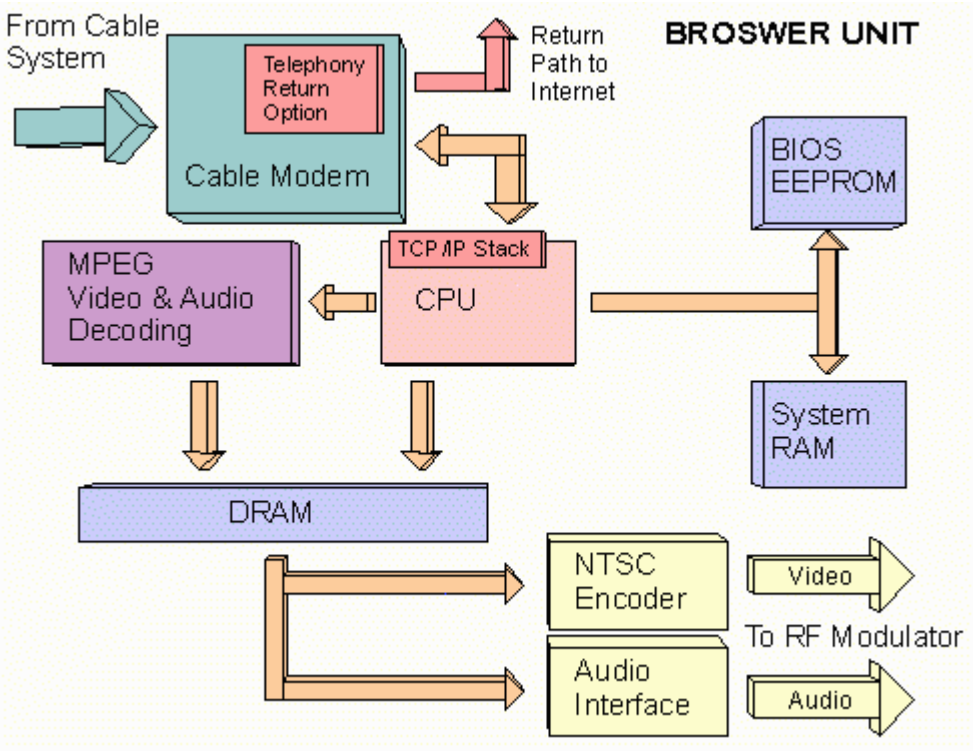


Figure 3.5: Option 1 Block Diagram of Decoder Components

In the second design, the sponsor wanted the CPU and the MPEG decoder to be addressable separately such that each would maintain a TCP/IP stack. The advantage of this design was that each subsystem would be addressable with the IP protocol and could be developed and tested separately. The disadvantage of this design was that all MPEG hardware decoders reviewed were designed for the sole purpose of decoding MPEG streams. Thus, a processor would need to be specified and software would be needed to perform both the MPEG decoding and the TCP/IP stack management. A block diagram of this scenario can be found in Figure 3.6.

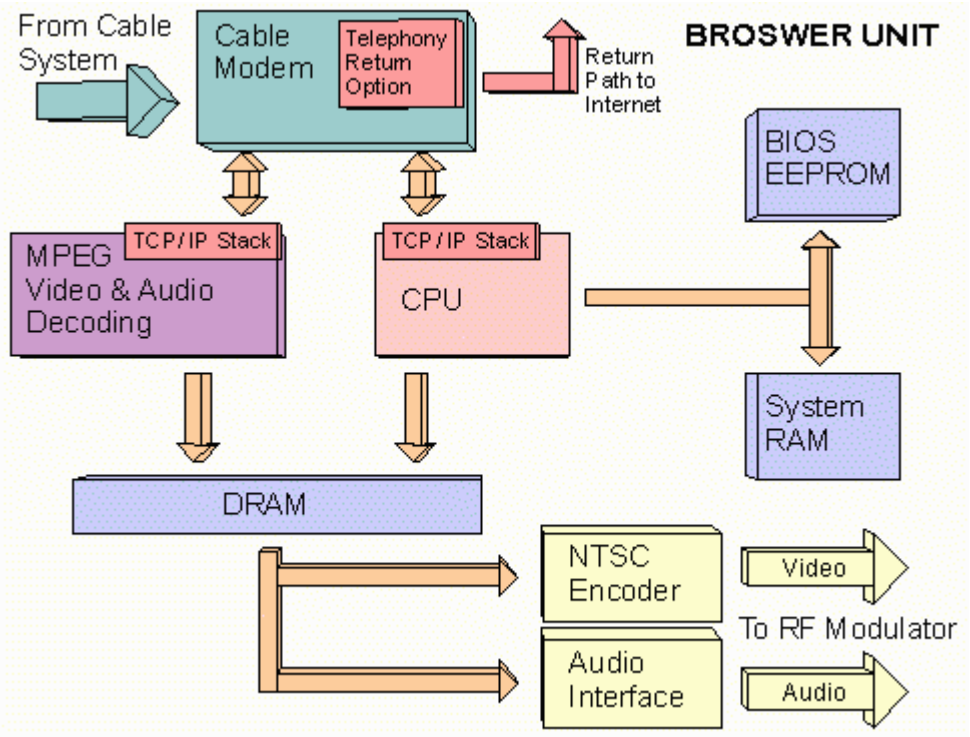


Figure 3.6: Option 2 Block Diagram of Decoder Components

Regardless of which design was chosen, a decision would need to be made as to how the graphical rendering of the web page would be accomplished. If the CPU had enough processing power, it could be tasked to perform rendering, or a VGA chip could be added to do the rendering.

The rendered web page data and output of the MPEG decoder would both serve as inputs to the NTSC encoder as seen in both block diagrams above. A user would be able to either view an MPEG movie or browse web pages. The benefit of adding MPEG decoding to this system would be providing users with the ability to view MPEG coded advertisements.

3.4.3 Design Concept 3 Conclusion

This Design Concept was only in the process of being specified when the sponsor expressed interest separating the Browser System into two pieces. Both potential designs described above showed promise of functioning as the sponsor desired. However, the sponsor felt that a distributed system would more a more efficient method of providing the IVDS Web Browsing service to more people. Simplifying the unit near the consumers home would reduce the cost of each unit. This would enable each unit to serve fewer users, effectively increasing each user's access to this system. At this point, our effort was directed toward this new Design Concept and the single unit IVDS WWW Browser System was disregarded.

3.5 Design Concept 4

3.5.1 System Description

In this Design Concept, the sponsor envisioned a slightly different system design than the previous Design Concepts. The Browser itself would no longer be one unit performing all tasks involved in browsing the web and displaying the web pages to the user. Instead, the system would be divided into two pieces, each performing one of the two above functions.

A powerful computer called the Browser would perform the functions of receiving and processing user messages, fetching requested web pages from the internet, and rendering the web page displays. The IVDS Repeaters would forward web session

messages to this large computer across the internet. This large machine would receive user messages from multiple Repeaters and serve many users. This machine would encode all of the rendered web page displays and audio into the MPEG format.

The Browser would then act as a server, forwarding the display data to smaller Decoder Units that would decode the MPEG and display the audio and video to the user. The cable system network would be used to transport the MPEG coded audio and video from the large Browser to the Decoder Units, as seen in Figure 3.7.

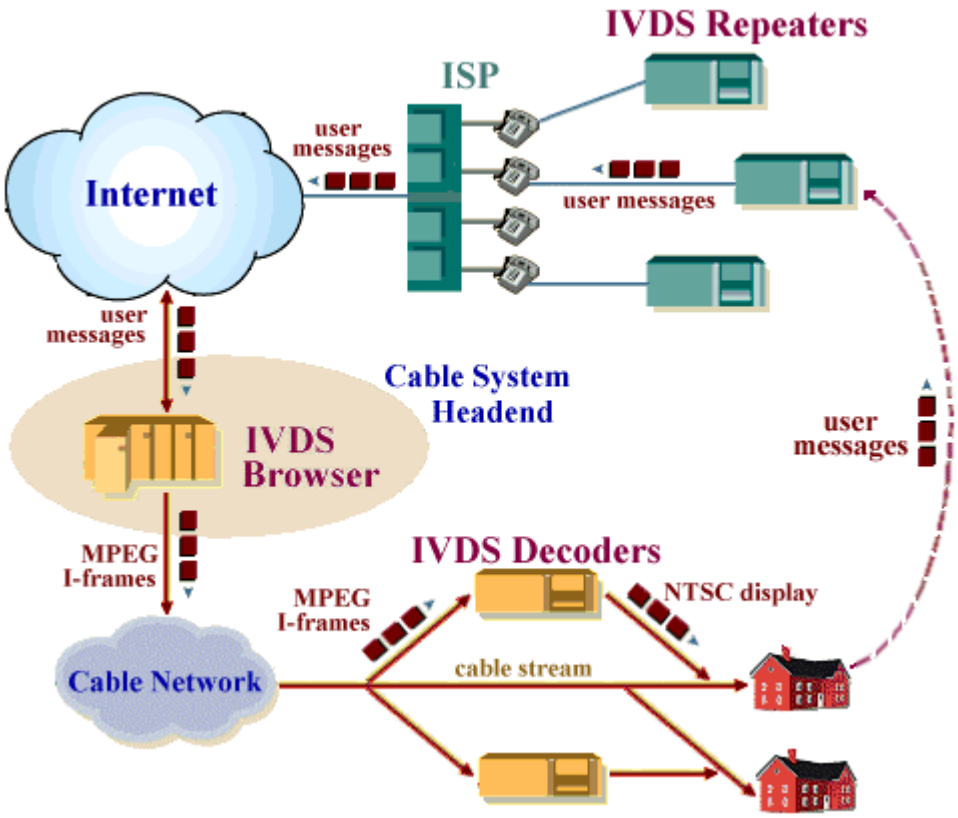


Figure 3.7: Design Concept 4 System Diagram

The Browser would reside at the cable system headend where it would have its own high speed connection to the internet. The Browser would receive user messages from the Repeaters and retrieve web pages using this internet connection. The cable

system network would serve as the physical path for transmission of the MPEG coded displays to the Decoder Units, as shown in Figure 3.7.

The Decoder Units would be responsible for receiving and decoding the MPEG coded data, converting the video and audio into an NTSC signal, and multiplexing the NTSC display signal into the cable stream so that it could be displayed on the user's television.

Each Decoder Unit would need a cable receiver in order to receive the MPEG encoded the display from the cable stream, as shown in Figure 3.8. A Decoder would not be able to transmit back to the Browser. It would simply receive the MPEG data and "play" it for the user. This would simplify the Decoder design, minimizing its cost.

Each unit would contain an MPEG decoder chip set for decoding the video and audio, an audio DAC for audio conversion from digital to analog, an NTSC encoder to encode the video into NTSC, and TV signal generator to combine the audio and NTSC video. Finally, another piece of hardware would be needed to shift the NTSC signal into the bandwidth of the channel allocated to the IVDS service.

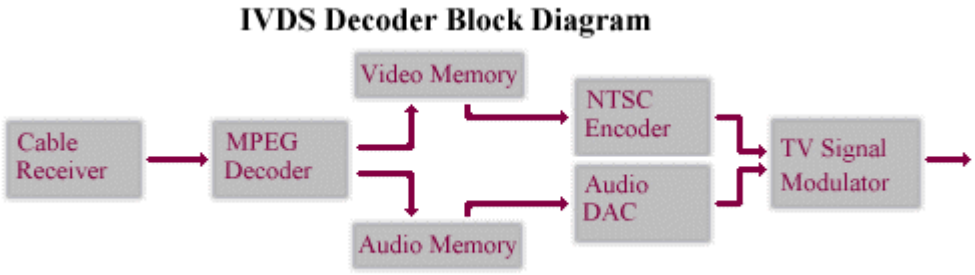


Figure 3.8: Design Concept 4 Decoder Block Diagram

3.5.2 MPEG Issues

Rendered web pages would be encoded into MPEG using I-frames only, thus reducing processing power required. The use of only I-frames would also reduce the decoding time and processing power required in the Decoder Units.

MPEG I-frames contain information about the pixels within image itself. MPEG B-frames and P-frames contain information used to perform motion prediction and compensation within a series of images.

The decoding of P-frames and B-frames relies on I-frames received both in the past and future. I-frames, while larger in byte size are much less complex to encode and decode while P- and B-frames are smaller in size and are much more computationally complex to encode and decode.

At that point in the research, the sponsor decided that we would not try to support animation of images. The IVDS Browser would encode web page displays into I-frames only because there would be no motion involved. Accordingly, the Decoder Unit would only have to decode I-frames. This would reduce delay and required CPU cycles at the decoding point.

To minimize bandwidth usage, the Browser would ideally only encode a new I-frame for transmission when an update would need to be made to the display the user was viewing. Suppose the user loads a new web page and begins reading the content of that web page. The Browser would encode and transmit a first I-frame for that 640x480 segment of the web page display.

The Browser would not send any more MPEG I-frames to that Decoder until the user interacted with that web page. Then the Browser would send a second I-frame to update the display according to the user's action.

This idea of non-continuous updates introduces two issues. MPEG encoders are normally used for encoding a continuous stream of video and audio. It was unknown if there would be a problem if the MPEG encoder were to be given a sporadic input.

In the MPEG Sequence Header, there is a four bit Frame Rate Code field allowing frame rates ranging from 23.976 to 60 frames per second [5]. This leads to the assumption that the MPEG decoder expects a fixed frame rate and that the encoder should encode at this frame rate. If this is the case, the sporadic updating of the display may not work as intended at the Browser and the Decoders.

If the Browser were to update the Decoders at 23.976 frames per second, the overall cable network bandwidth required by the IVDS System would increase significantly compared with the bandwidth required by the sporatic updating scheme. The amount of bandwidth that the IVDS System could use would be determined by business agreements between IRS, Inc. and each individual cable service provider.

3.5.3 Design Concept 4 Conclusion

Concerns about the MPEG issues involved in this Design Concept ultimately led the sponsor to remove the MPEG transmission format as the method to transport audio and video from the Browser to the many Decoder Units. The sponsor directed the project focus on a new Design Concept using Ethernet as the transmission method used to

transport the rendered web page data and audio to the Decoder Units. This became the final design and its concept will be presented in the next chapter.

3.6 Conclusion

This chapter has discussed each of the succeeding IVDS Browser System Design Concepts. Each system was introduced and the relevant hardware issues were presented. In each of the four Design Concepts, the issues involved or a new application motivated the sponsor to direct our efforts to the next Design Concept. The research into each of these Design Concepts was beneficial to the development of the final Browser System presented in the next chapter.

Chapter 4. The Final IVDS Browser System

4.1 Introduction

In the final design concept, the IVDS Browser System is divided into two physical entities, a large IVDS Browser computer and smaller IVDS Decoder Units, as shown in Figure 4.1. The large Browser computer is responsible for receiving user messages from multiple IVDS repeaters, processing user messages, retrieving and caching web pages from the internet, rendering the web pages, and transmitting display and audio data to the Decoder Units.

The Decoder Units will store the display data, encode the audio and video into NTSC, and multiplex the NTSC signal into the cable stream. This display will then be available on the "web channel" of the user's television.

A user will then use the Audio Link to interact with the web page on the television. User actions will be transmitted to an IVDS Repeater, where they will be forwarded on to the Browser and processed accordingly.

The Browser will reside at the cable system headend where it will have a high speed connection to the internet. It will receive user messages from the IVDS Repeaters as well as retrieve requested web pages using this internet connection. The Browser will manage user commands as well as manage transmissions to the Decoder Units from the active web browsing sessions. Rendered web page display data as well as associated audio data will be transmitted within an IVDS and Ethernet packet format through the cable system network to the Decoder Units.

The sponsor felt that the effort required to resolve and potentially work around the MPEG issues was not worth the system features that MPEG provided. Therefore, the use of MPEG was removed from the system entirely. It was determined that Ethernet packets would be used to carry display information from the Browser to the Decoders.

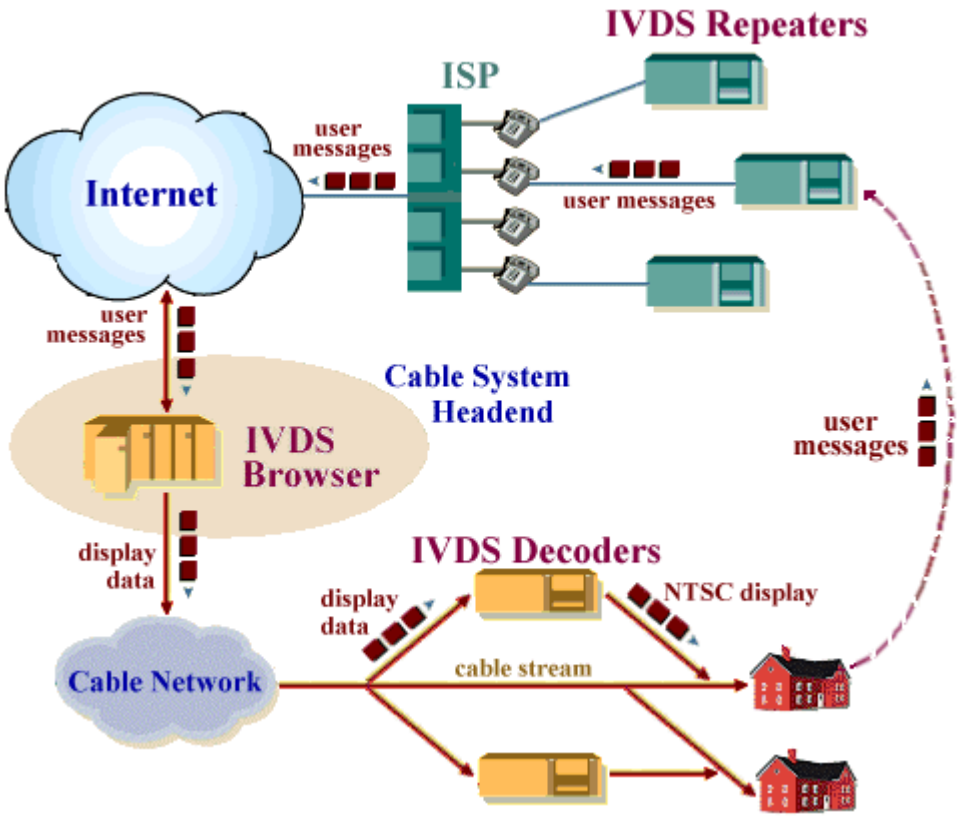


Figure 4.1: Final IVDS Browser System Design

Audio and rendered web page data will be encapsulated within an IVDS frame format before being presented to the lower layer Ethernet for packetization. The IVDS frame format will provide information about the type of display data that a Decoder will receive and what actions to take to process that data. The specification of the IVDS frame format will be covered in the next chapter.

Each Decoder Unit will use a cable modem to receive data from the cable system network. The Decoder Unit will store the rendered images and audio data in RAM, encode the image data into NTSC, and embed the audio into the NTSC audio subcarrier, and multiplex the NTSC audio and video signal into the cable stream. The display will then be viewable on televisions that the Decoder Unit serves. Users will use the Audio Link as the wireless return path to the Browser to interact with the web pages presented on their televisions.

One of the benefits of dividing the system into two components is the reduced cost of the Decoder Units. The sponsor felt that if the Decoder Unit cost could be reduced, that it would be possible to dedicate a Decoder Unit to each home. The visibility of each browsing session would then be limited to the televisions within that home. This would provide more privacy for the users as well as access to the IVDS WWW Browsing Service independent of other users.

4.2 Browser to Decoder Communication

Two network packet formats will be used to transmit the audio and video display data from the Browser to the Decoders. The lowest layer packet format will be the Ethernet packet format. At the next higher layer, a special IVDS packet format will be used to encapsulate the audio samples and RGB image data.

The Browser will render the web page images in an RGB format to be encapsulated in IVDS packets. Audio samples will be forwarded to the Decoders in a 16-

bit sample format. The IVDS packets will in turn be encapsulated in Ethernet packets. The specification of the IVDS packet format will be presented in the next chapter.

The Browser will send IVDS packetized audio samples and video display data across the cable network in Ethernet packets. Each decoder unit will contain a 100-Mbps Ethernet chip set to receive the Ethernet packets from the cable modem. The bandwidth allocated for transmissions from the Browser to the Decoder will be determined by arrangements between IRS, Inc. and each individual cable provider.

No return path will be provided for the Decoder Units to communicate with the Browser. The Decoder Unit's sole responsibilities will be to receive data and display it to the user. This simplex communications channel does not require the cable provider's network to have been upgraded to support bi-directional communications.

Within the bandwidth allocated by the cable provider, the Browser will be the only transmitter. The Decoders function as receivers only. The Browser will be able to support the web browsing sessions of many users. This virtual Ethernet network will have only one transmitter, so there will be no packet collisions. This will allow more efficient use of the allocated cable network bandwidth as well as simplified MAC processing requirements of the Decoders.

The sponsor felt that the 10-Mbps that standard Ethernet provided would not be sufficient to support the large number of users at peak usage times. Therefore, he specified that the Browser and Decoder Units use 100-Mbps Ethernet chip sets so that data rates exceeding 10-Mbps could be achieved on the cable network.

The sponsor also felt that the usage of the 100-Mbps Ethernet technology would make the system attractive for future applications in which a true 100-Mbps Ethernet network could be used.

While current cable modems only support bandwidths up to roughly 40-Mbps, the sponsor felt that the IVDS Browser System should provide support for higher data rates that could be acheived in the future. When the data rates the cable modems supported had increased, the ability would then exist to upgrade the IVDS Browser System cable modems. This upgrade would provide additional bandwidth that could be used to support more users.

4.3 Ethernet Packet Issues

Ethernet packets will be used to transport the IVDS packetized display data to the Decoders. The Decoders will use both the IVDS and Ethernet packet header information to process the display information. However, the Decoders will not use the CRC field of the Ethernet packet.

When a Decoder receives a corrupt Ethernet packet, it will not be able to detect errors in that packet and will attempt to process that packet as normal. No CRC verification will be performed by the Decoders. There is no return path for retransmission requests, as well as no higher layer protocol support within a Decoder Unit to make that request.

This decision to ignore the CRC was made for two reasons. First, verifying the CRC will require adding complexity to a Decoder. One of the goals of the separation of

the Browser System into two pieces was to minimize the functional responsibility placed on the Decoder Unit. The fewer tasks that a Decoder must perform, the less the requirement for powerful hardware, and the lower the unit cost of a Decoder Unit.

Secondly, the sponsor felt that if all the display data encapsulated within one Ethernet packet were corrupt, it would have a minimal effect on the audio and video presented to the user. The sponsor felt that in the worst case, when the corrupt data became a problem to the user, he or she would press the button to effectively reload the page as one would do using in a normal web session on a PC. This reload would cause the Browser to resend all of the display data to the Decoder. The user would become in effect the higher layer processing that initiates a retransmission request from the Browser.

4.4 IVDS Browser Issues

In order to support large numbers of users, the IVDS Browser will be a powerful computer capable of supporting the browsing activities of many users. The Browser machine will also need to be scaleable so that more computing power can be added as the number of users that the Browser services increases.

Using a large machine to process all of the users' browsing sessions will allow caching of commonly viewed web pages. This will reduce the number of fetches from the internet as the most frequently viewed web pages can be retrieved from a common cache for any user. The smaller Browsers in the previous design concepts had minimal memory spaces and were only capable of caching the web page the user was currently viewing.

When a user initiates a web browsing session, he or she begins at a common starting point. The Audio Link does not have a keyboard and does not allow users to directly enter URLs. Instead, each user must be follow a series of links to arrive at a desired web page. If a user frequently visits a particular page, he or she must always follow a series of links to arrive at that web page from the starting point.

The Browser will allow users to maintain bookmarks to allow them direct access to frequently visited web pages. The smaller Browsers in the previous design concepts had no hard disks and were not capable of providing the bookmark function.

The sponsor was investigating machines for use as Browsers made by Tandem, Sun, SGI, and IBM. John Stanhope, a member of the research group, was directed to look at the requirements of Browser, as well as to begin the design of the software to retrieve, process, and render the web pages themselves.

Theodoros David, another member of the research group, was tasked to investigate the cable network issues and software to interface to the Ethernet transmission module. Theodoros was also responsible for specifying the Ethernet chip set to be part of each Decoder Unit. His research is presented in his thesis entitled *Networking Requirements and Solutions for a TV WWW Browser* [19].

4.5 Conclusion

This chapter has provided an introduction to the final distributed IVDS World Wide Web Browser System architecture. This design involves a distributed system in which from a large central Browser performs the browsing function for many users and

forwards the display data to many small Decoder Units. This display data is encapsulated using both the special IVDS frame format and Ethernet packet. The packetized display data can be transported by either a cable system network or a 100-Mbps Ethernet network.

The motivation for the usage of the IVDS packet format was presented as well as the decision not to use the CRC field of the Ethernet packet. The next chapter will present a board-level description of the IVDS Decoder and a discussion of some of the features that the IVDS packet format brings to the IVDS Browser System.

Chapter 5. The IVDS Decoder Unit

5.1 Introduction

The IVDS Decoder Unit is responsible for receiving the display data from the Browser and displaying it to the user over the cable stream into his or her home. The audio and image display data received by the Decoder will be encapsulated in a special IVDS packet format. This packet contains header fields which convey information about the packetized display data. The IVDS packets will be encapsulated within Ethernet packets transmitted from the Browser to the Decoders.

The Decoder will use a 100-Mbps Ethernet chip set to receive the Ethernet packets. Ethernet packets will be forwarded to an FPGA (Field Programmable Gate Array) which will perform Ethernet and IVDS packet processing and memory management. The FPGA will continuously refresh the NTSC display using the pixels stored in memory. Additionally, the FPGA will forward the stored audio samples to an audio DAC within the Decoder which will convert the samples into an analog signal. The internal FPGA design will be discussed in detail in the next chapter.

Three distinct memory units will be used to store the 24-bit RGB pixels and the 16-bit audio samples. Each memory unit will be discussed later in this chapter. An audio DAC will be used to convert the audio samples into an analog signal. A NTSC encoder will convert the 24-bit pixels into a format viewable on a television. A composite video generator will combine the audio and NTSC signal and generate a continuous signal to modulate into the cable stream. Figure 5.1 shows the components that make up the Decoder.

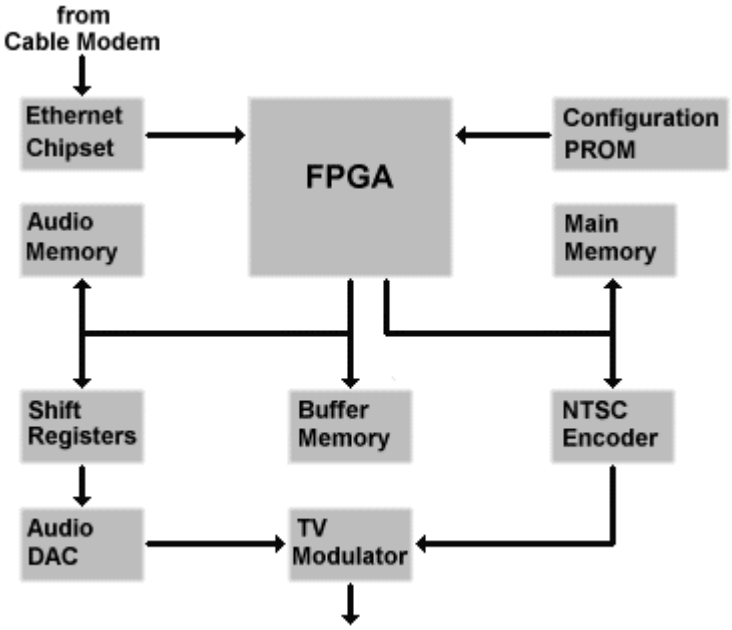


Figure 5.1: IVDS Decoder Block Diagram

5.2 Ethernet Chip Set

Theodoros David was responsible for specifying the 100-Mbps Ethernet chip set. He compared various chip sets on the basis of cost, functionality, and number of chips in the chip set. The sponsor wanted to minimize the number of integrated circuits in the Decoder. The sponsor ultimately approved the National Semiconductor DP83840 10/100 Mb/s Ethernet Physical Layer Chip set.

The Ethernet chip set is designed to interface to another chip that performs MAC layer functions. This interface is refered to as the MII (Media Independent Interface). The IVDS system does not require MAC layer processing, so the Ethernet chip set will interface directly to the FPGA instead using a subset of the lines provided by the MII.

While the Ethernet chip set will receive data at up to 100-Mbps, it uses four parallel lines to forward data to the FPGA. Data is sent over these four lines at 25-Mbps.

The Ethernet chip set uses the additional CRS, DV, and ER lines to send control information and a 25MHz clock to the FPGA. Figure 5.2 shows the interface between the Ethernet chip set and the FPGA.

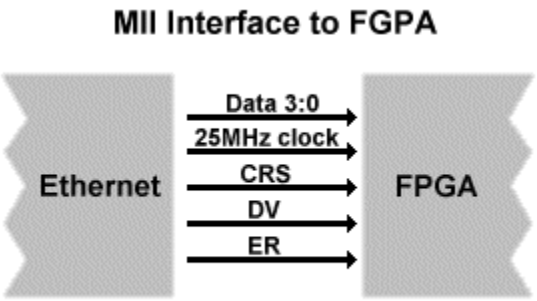


Figure 5.2: Ethernet chip set interface to the FPGA

5.3 FPGA

The Xilinx XC5210-4PQ208 FPGA is specified for this design by Theodorous David. Some of the pin and gate statistics of the FPGA can be found in Table 5.1. The FPGA will perform two main functions in the Decoder. This FPGA is the fastest available in this series as denoted by the "-4." Table 5.2 shows some of the expected performance for several common circuit functions.

The functions of the FPGA design can be categorized into two areas. First, the FPGA will perform Ethernet and IVDS packet processing. The packet processing will determine how the FPGA manages the actual display data.

Table 5.1 : Characteristics of the XC5210-PQ-208 [20]

Total Number of Pins	208
Number of I/O Pins	164
Max Logic Gates	16,000
Typical Gate Ranges	10,000 - 16,000

Table 5.2 : Performance for Several Common Circuit Functions [20]

Function	XC5200 Speed Grade			
	-6	-5	-4	-3
16-bit Decoder from Input Pad	9 ns	8 ns	7 ns	6 ns
24-bit Accumulator	32 MHz	39 MHz	45 MHz	50 MHz
16-to-1 Multiplexer	16 ns	13 ns	11 ns	9 ns
16-bit Unidirectional Loadable Counter	40 MHz	50 MHz	59 MHz	65 MHz
16-bit U/D Counter	40 MHz	50 MHz	59 MHz	65 MHz
16-bit Adder	24 ns	20 ns	17 ns	15 ns
24-bit Loadable U/D Counter	36 MHz	42 MHz	48 MHz	52 MHz
	Preliminary			Advance

The second responsibilty of the FPGA is the memory management. The incoming display data must be stored in the proper memory locations. Furthermore, the pixels must be copied by the FPGA in a synchronous way from the Buffer Memory to the Main Memory. Finally, the FPGA must update the audio DAC a regular intervals so that the audio samples can be “played” to the user.

The logic design that will be loaded into the FPGA will be presented in detail in the next chapter.

5.4 Serial Configuration PROM

The Xilinx XC17256D(SCP) Serial Configuration PROM will be used to program the FPGA. The configuration of the FPGA is volatile. Each time power is first applied to the Decoder, the FPGA must be configured. This PROM will be programmed to contain the information to configure the FPGA to function properly in the Decoder.

5.5 Shift Registers

Sixteen-bit sound samples are input to a pair of 74LS165 shift registers. These shift registers are loaded with audio samples read from the Audio Memory. Each 16-bit audio sample is then shifted into the 1-bit input of the audio DAC.

5.6 Audio DAC

The Burr-Brown PCM56U DAC accepts a 1-bit data input, latches 16-bit audio samples from this input, and converts the digital samples into a analog output. A board level diagram of the Decoder Audio Circuit can be found in Appendix D.

5.7 NTSC Encoder

The 24-bit RGB color pixels will be converted into the NTSC format using the Texas Instruments TMS320AV420 Digital NTSC Encoder. This encoder converts RGB pixels to the analog S-video signal using digital signal processing techniques. Pixels are input to the encoder using a 13.5MHz clock.

5.8 Composite Video Generator

The output of the NTSC Encoder serves as the video input to the Motorola MC1374 TV Modulator Circuit. The analog audio signal from the Audio DAC serves as the audio input to this chip. This chip combines the audio and video signal and generates a continuous NTSC signal which can be modulated into the cable stream.

5.9 Ethernet and IVDS Packet Processing

Each Decoder Unit will use a cable modem to provide the physical interface to the cable network. The cable modem will receive Ethernet packets from the cable network and forward them to the 100-Mbps Ethernet physical layer chip set on the Decoder itself. The cable modem specified for use in this system will have to be able to interface to the 100-Mbps Ethernet chip set.

Normally, on an Ethernet network multiple transmitters contend for bandwidth and packet collisions result. These packet collisions are resolved at the Ethernet MAC layer. In the IVDS Broswer System however, no MAC layer processing is needed. The Browser is the only transmitter on the network so there are no packet collisions to resolve.

All Ethernet packets sent from the Browser will be received by the Ethernet physical layer chip set in every Decoder and forwarded to the on-board FPGA. This FPGA is the processing engine of the Decoder. The FPGA will receive the Ethernet packets and look at the Destination Address field of the Ethernet packet to verify that the packet is intended for that Decoder. Each Decoder has a unique address that it matches against the address in the Destination Address field. Every packet is received by every Decoder, however, each Decoder keeps only the packets intended for it. All other Decoders will ignore packets not intended for them.

The Browser will serve many fewer Decoders than it could possibly address in the 48-bit Destination Address field, so there are bits in that field that will not be used to address the Decoders. A special bit is reserved within the Destination Address field of the Ethernet packets that represents a broadcast flag. If the Browser sets this bit in an

Ethernet packet, every Decoder Unit will recognize that the packet is being broadcast and process that Ethernet packet.

This broadcast mode will allow the Browser to send a common message to every user that is currently using the system. How the FPGA will use each Ethernet field is discussed below in the Ethernet Packet Specification.

When a Decoder receives a packet that is intended for it, that packet will be accepted and the remaining header fields and data in the Ethernet packet will be processed by the FPGA. The Ethernet data field may contain one or more IVDS packets. The actual data that represents the audio and video information is encapsulated within IVDS packets. The IVDS packet header fields will allow the FPGA to determine what type of display data it is receiving and how to process that information. The specification for the IVDS packet format is discussed below in the IVDS Packet Specification. Additionally, Figure 5.3 illustrates the Ethernet and IVDS packet formats.

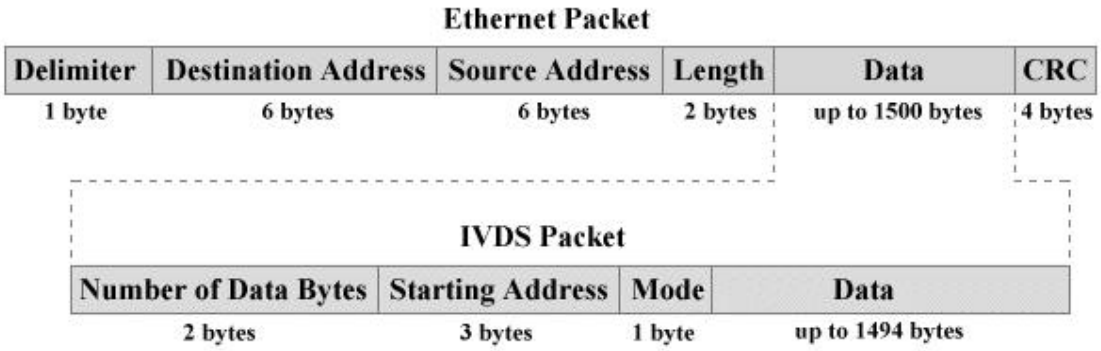


Figure 5.3: Ethernet and IVDS Packet Formats

5.10 Ethernet Packet Specification

5.10.1 Delimiter (1 byte)

The Delimiter is at the head of the Ethernet packet and is a constant value of 10101011 which will be detected by Decoder Unit and the packet will be recognized as a the head of an Ethernet packet [21].

5.10.2 Destination Address (4 bytes)

The first 10 bits will contain the Decoder Unit address. If the eleventh bit is set to a logic high, the Browser is in broadcast mode and the packet is intended for all Decoder Units. The remaining 37 bits of the Destination Address are ignored.

5.10.3 Source Address (4 bytes)

All 48 bits are ignored and not used by the Decoder Unit.

5.10.4 Length (2 bytes)

The Length field specifies the number of bytes in the Data Field of the Ethernet packet.

5.10.5 Data Field (up to 1500 bytes)

The Data Field contains up to 1500 bytes and may contain multiple IVDS Packets containing the audio and pixel display data. (see Section 5.11 *IVDS Data Packet Specification* for more details.)

5.10.6 CRC (4 bytes)

The CRC is ignored and not verified by the Decoder Unit.

5.11 IVDS Data Packet Specification

5.11.1 Data Length (2 bytes)

This field specifies the number of bytes of data in the data field of the IVDS packet. A counter within the FPGA will be used to count down from this value to zero. When the counter reaches zero, this signifies the end of the IVDS Data Packet. At this point the Decoder Unit waits for a new IVDS Data Packet which is part of the current Ethernet Packet.

The end of the IVDS packet may also be the end Ethernet Packet. This will be determined by a counter loaded from the Ethernet length field. When the end of the Ethernet packet is encountered, the Decoder Unit will wait for a new Ethernet packet to arrive. Multiple IVDS packets can be sent within the data field of a single Ethernet packet.

5.11.2 Starting Address (3 bytes)

This field specifies the address in Main Memory in which the first pixel color in the data field will be written to. If the following data is a color palette to store, this field contains the address in the Buffer Memory in which the first palette entry will be written into. The Browser is responsible for sending an address within the memory space allocated to the palettes.

The starting address value will be loaded into two counters, independent of the mode. If the mode is set to the 8-bit or 24-bit pixel modes, only one of the counters will be used. As each new color pixel value is received, the counter will increment the memory address used to store the pixel data. If in the color palette mode, the other counter will be used and the following 256 24-bit palette entries will be written into the Buffer Memory using addresses provided by this second counter. If the Decoder is set to an audio mode, neither counter is used.

Each pixel color values in the IVDS packet will be sequentially written into a memory space beginning at the starting address and ending at the starting address plus the Data Length field value. Each new non-sequential address that needs to be updated will need to be sent in another IVDS Data Packet that may be held within the 1500 bytes of the Ethernet Data Field of the current Ethernet Packet or in another Ethernet Packet. For example, if the Browser wanted to send the color pixel value for memory locations x and x+2, each pixel value will have to be in different IVDS packets.

5.11.3 Mode (1 byte)

This field specifies what type of data the following data field of the IVDS packet contains. (see Section 5.12 Mode Specfication for further details)

5.11.4 Data Field (up to 1494 bytes)

The data field can contain up to 1494 bytes of data. The data within this field can be 8-bit color pixel data, 24-bit color pixel data, 16-bit audio samples, or 24-bit color

palette entries. The data field of the IVDS Data Packet must contain only one type of data, as specified in the preceding Mode Field.

If a new type of data is to be sent, a new IVDS Data Packet must be generated and sent. More than one IVDS Data Packet can be sent within a single Ethernet Packet. However, IVDS Data Packets can not be continued across Ethernet Packets. At the end of the Ethernet data field, the IVDS Data Packet must end.

5.12 Mode Specification

Table 5.3 shows the value for each bit in each of possible modes. Bits four, five, and six specify the mode to operate in. Each mode represents how the FPGA should process information in the Data Field of the IVDS packet.

Bit zero through bit three are only used in the 8-bit color mode to specify which color palette to use to convert the 8-bit pixel values to 24-bit values. In all other modes bit zero through bit three are not used.

Bit seven is not used by the FPGA.

Table 5.3: Mode Field Bits

Bit Number								Mode
7	6	5	4	3	2	1	0	
x	0	0	0	x	x	x	x	24-bit color
x	0	0	1	p	p	p	p	8-bit color
x	0	1	0	x	x	x	x	new audio samples
x	0	1	1	x	x	x	x	continuing audio samples
x	1	0	0	x	x	x	x	load palette

x = not used
p = used to specify which palette to use to convert the incoming 8-bit color pixels to 24-bit color pixels

5.13 Audio Memory

The Decoder contains three distinct memory units used to buffer and store the received audio samples and image data. The Audio Memory specified is a NEC µPD4218160 1Mx16 DRAM. The 16-bit audio samples received by the FPGA will be stored in this Audio Memory. The Audio Memory is large enough to store roughly 45 seconds of audio data.

$$\frac{1x10^6\,samples}{22.1x10^3\,samples/\sec ond} = 45.2\sec onds$$

The audio samples stored in the Audio Memory are loaded at 22.1 KHz rate into the shift registers that shift the audio samples into the audio DAC. The resulting analog audio signal will be combined with the output of the NTSC encoder and then modulated into the cable stream.

The FPGA contains two internal counters involved in Audio Memory management. An input counter will determine the address in the Audio Memory where the incoming audio sample is to be stored. The Browser is responsible for sending only enough information to fill up the Audio Memory. Once the Audio Memory is full, the Browser must wait until at least some of the audio samples have been forwarded to the Audio DAC. If the Broswer sends too many audio samples too quickly the FPGA will overwrite audio samples that have not been forwarded to the audio DAC.

An output counter is implemented within the FPGA to determine the Audio Memory address the next audio sample will be read from. Each 16-bit audio sample will be loaded into a pair of 8-bit shift registers. These shift registers will shift the 16-bits into

the audio DAC for conversion into an analog audio signal. These shift registers are board-level components external to the FPGA.

The Audio Memory counters are setup such that the output counter can not pass the input counter value. Suppose the Audio Memory is full of audio samples that had already been played. The Decoder begins receiving a new set of audio samples and begins storing them at the beginning of the Audio Memory. The output counter will begin playing from the first location in the Audio Memory.

If its address count passes that of the input counter, the output counter will begin playing old audio samples. Therefore the output counter is only allowed to increment while its address count is not equal to the input counter. When the two counter values are equal, the output counter will stop incrementing until the input counter receives more audio samples and begins incrementing its count to store them. This mechanism will allow various length audio samples to be "played" by the Decoder without concern about deleting old audio samples from the Audio Memory.

5.14 Main Memory

The 24-bit 640x480 RGB pixel representation of a web page will be stored in a series of three NEC μPD424800 512Kx8 DRAMs. These DRAMs together form the 512x24 Main Memory of the Decoder. The FPGA will implement an internal counter to maintain the visual display by cycling through Main Memory addresses and performing reads from this memory every 74.074ns. Each 24-bit RGB pixel read from the Main Memory will be latched by the NTSC Encoder pixels to refresh the display.

The Browser can send either 8-bit or 24-bit RGB pixels to the Decoder. NTSC encoders designed to accept the RGB pixel format as an input require that the RGB data be 24-bits. The Decoder must convert all 8-bit pixels into 24-bits before storing them in the Main Memory.

5.15 Buffer Memory

The final piece of memory in the Decoder is a series of six Hitachi HM62832UH 32Kx8 SRAMs. These SRAMs together form the 32Kx48 Buffer Memory of the Decoder. The Buffer Memory is intended to temporarily store newly received 24-bit RGB pixels as well as 24-bit entries in the color palette lookup tables. Stored with each 24-bit pixel in the Buffer Memory is a 24-bit address that represents the intended location of the pixel in the Main Memory.

The 640x480 image stored in the Main Memory requires 307,200 pixels. Each pixel received by the FPGA may need to be placed in any location of the Main Memory. Therefore each new pixel is stored with a corresponding 24-bit address that represents where the pixel should be stored in the Main Memory. The 24-bit pixel and 24-bit address are stored in a 1x48 memory slice of the Buffer Memory. The Buffer Memory will be able to store 28,000 of these pixel-address pairs. The remaining 4,000 1x48 memory slices will be used to store 24-bit color palettes entries.

A pixel stored in the Buffer Memory will be copied into the Main Memory when the corresponding 24-bit address stored with it matches the current Main Memory address specified by a counter within the FPGA. For example, a pixel in the Buffer Memory has a

corresponding address location *X*. A counter in the FPGA is cycling through the memory locations of the Main Memory, 24-bit pixel values are being read, and then latched by the NTSC Encoder.

When the counter reaches memory location *X* of the Main Memory, rather than performing a read from the Main Memory, the FPGA will perform a write to the Main Memory using the pixel from the Buffer Memory. During this cycle, the NTSC Encoder will latch this new pixel and use it to update the display. This new pixel will now be in the correct location in the Main Memory. Figure 5.4 illustrates this example.

There is an important design tradeoff in the size of the Buffer Memory. The number of 24-bit color pixels that the Buffer Memory can store is inversely proportional to the time it will take to refresh entire screen of 307,200 pixels. The Buffer Memory was initially designed to store up to 28K pixels.

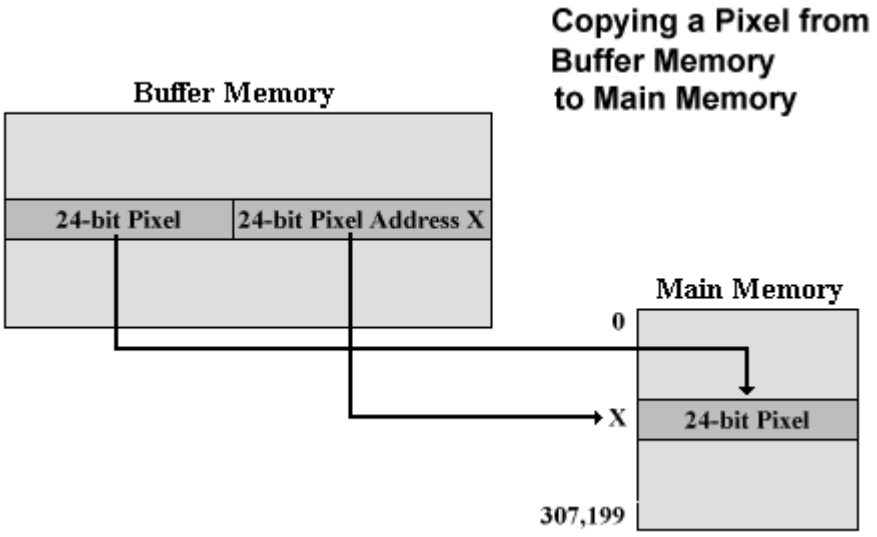


Figure 5.4: Moving a Pixel from Buffer Memory to the Main Memory

Using this design, 0.4 seconds are required to refresh the entire screen. This delay is introduced by the temporary storing of new pixels and the necessary wait for them to be copied into the Main Memory. Each time the Browser sends enough pixels to fill up the Buffer Memory, it must wait until all of the pixels have been copied into the Main Memory before it can transmit more pixels. Calculations are provided in the Appendix A that show this result.

As a result of this delay, the Decoder design was modified so that another series of six 32kx8 SRAMs may be added to the board doubling the size of the Buffer Memory. A jumper on the board will be used to notify the FPGA that another series of SRAMs have been added to the board. The FPGA can then use both groups of SRAMs to temporarily store pixels.

This increase in the size of the Buffer Memory would nearly double the memory space in which newly received pixels could be stored. The delay in transmitting an entire screen of pixels from the Browser to the Decoder would be reduced to 0.2 seconds. The calculations in Appendix A show the effect this extra memory would have on the time required to update the screen.

5.16 Benefits of the IVDS Packet

While the use of the IVDS frame format to transmit display data adds overhead, it provides several beneficial features to the IVDS Browser System. The Browser will be able to send both 8-bit and 24-bit color pixels to the Decoders. The playing of audio will

be able to be stopped upon leaving a web page. Furthermore, the Browser will be able to update a portion of the screen rather than always refreshing the entire screen.

5.16.1 8-bit to 24-bit Conversion Using Color Palettes

The sponsor wants the Browser to be able to send both 8-bit and 24-bit color to the Decoders. However, the NTSC encoder in the Decoder requires a 24-bit input. Therefore, if 8-bit color data is sent to the Decoder, it will have to be converted to 24-bit color before given to the NTSC Encoder. This conversion will require a palette of colors to obtain a 24-bit color given an 8-bit color input.

Additionally, the sponsor felt it is necessary to use different color lookup tables at different times to present colors most effectively to the user. So the Browser is able to send sixteen different color palettes to each Decoder, where they are stored in the Buffer Memory and used when 8-bit color data is received by a Decoder Unit.

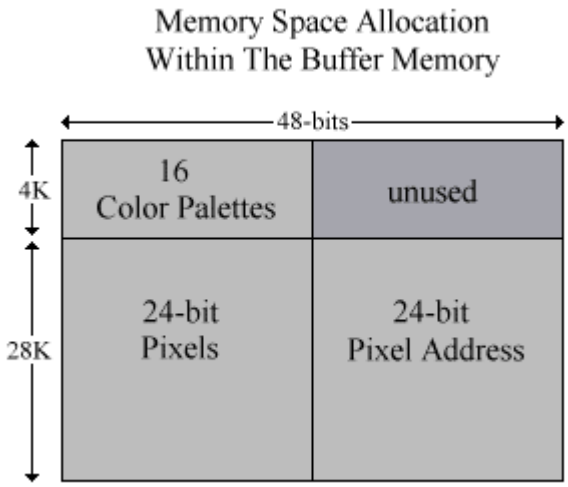


Figure 5.5: Memory Space Allocation Within the Buffer Memory

The palettes will be stored in the first 4k of 48-bit slices in the Buffer Memory. In order to simplify the FPGA design, each 24-bit palette entry is stored in the first 24-bits of each 48-bit slice. No information is stored in the remaining upper 24-bits of the 4k palette memory space. Figure 5.5 illustrates the space allocation within the Buffer Memory.

In order to achieve this functionality, a method of notifying a Decoder Unit of the type of data it is receiving will be necessary. The Decoder could receive 24-bit color data, 8-bit color data, a new color palette to store, or a series of 16-bit audio samples from the Browser. When the Browser sends 8-bit image data to a Decoder Unit, the Decoder will need to know which color palette to use to convert the 8-bit data into 24-bit data. So when 8-bit image data will be sent, an indication of the palette to use will need to be sent also.

The FPGA will convert the incoming 8-bit pixels into 24-bits before storing them in the Buffer Memory. Four bits in the Mode field of the IVDS packet header will tell the Decoder which palette to use. These four bits will allow usage of the sixteen different palettes the Decoder Buffer Memory is capable of storing.

The 4-bit palette and the 8-bit pixel itself will be used to address the palette memory space within the Buffer Memory. The resulting 24-bit value read out will be the 24-bit representation of the 8-bit pixel used as the address. This 24-bit value will be buffered for one clock cycle and stored in the Buffer Memory with its corresponding address.

The Mode field within the IVDS packet specifies the type of display data to follow in the Data field. The display data mode will last for the duration of the IVDS packet.

The type of display data sent within an IVDS packet must remain constant. If a new type of display data needs to be sent, a new IVDS packet must be sent. The next IVDS packet must specify the mode again.

5.16.2 Audio Mode

A Decoder Unit will be able to store a roughly 45 seconds of audio data. The Browser will be able to send this audio data much faster than the Decoder Unit will "play" it. Suppose the user is on a web page that has a looping sound sample embedded in that page. Then the user moves to another web page. The user will want the old sound bite to stop when he moves to the new web page and if present, the new sound bite to start. The Browser will need a method to send the Decoder Unit a new series of audio samples and force the Decoder Unit to overwrite the current audio samples that it was playing.

The FPGA will use the Mode field to determine whether the incoming audio samples are a new series or part of a continuing sequence. New audio samples will now be written from the beginning of the Audio Memory overwriting samples previously stored and audio samples will be read from the beginning of the Audio Memory. If the incoming IVDS frame contains a continuing sequence of audio samples the FPGA will not reset the counters and the samples will be stored normally. Previously stored audio samples will be read normally as well.

5.16.3 Partially Updating the Display

Consider the situation where the focus is on the first link on a web page. This focus will be indicated visually to the user. Suppose the user then moves the focus to the next link on the same web page. The screen will need to be updated to show that the focus has moved from the first link to the second link. This could be accomplished by updating the entire screen. However, it will be more efficient if only the portion of the screen that the first and second links occupied could be changed.

The IVDS frame format has been designed to provide the partial update of the display in the following way. The 640 x 480 display contains a total of 307,200 pixels. The Decoder Unit will store each pixel sequentially in Main Memory. Consideration must be made by the Browser to send void pixels to place in the Main Memory locations that will be read during horizontal and vertical retrace periods.

The value of the Starting Address field will be loaded into an internal counter of the FPGA. This counter will be incremented each time a new pixel is received and its value will be stored in the Buffer Memory in the same 48-bit slice as the pixel that it corresponds to. Each new color pixel received in the IVDS data field will be stored in the subsequent Buffer Memory location overwriting the previous pixel values.

If the Browser wishes to update a series of pixels from one line of the screen and a series of pixels from another line on the screen, it must send them in two different packets. Once these pixels are moved from the Buffer Memory into the Main Memory the screen will reflect the new updated area. A small rectangular portion of the screen could be

updated using several IVDS packets with the pixels of each containing a small horizontal strip of the screen.

Small portions of the screen could be updated as described above providing the additional feature of efficiently supporting animated images. The animated image could be processed and efficiently updated on the user's display without requiring the Browser to send the Decoder Unit an entire screenful of pixels.

5.17 Conclusion

This chapter introduced the IVDS Decoder Unit at the board level. Each of the major components of the Decoder were presented. A specification was provided that discussed how the FPGA will process each of the data fields the IVDS and Ethernet packets. A specification was provided for the Mode field of the IVDS packet format. The different units of memory within the Decoder were introduced and their function discussed. The benefits that the IVDS Packet provide to the IVDS Browser System were presented. The next chapter will describe the internal design to be used inside the FPGA.

Chapter 6. Block Level Design of the FPGA

6.1 Introduction

The FPGA within the IVDS Decoder Unit coordinates all of the packet processing and memory management functions that take place within the Decoder. Data packets are received by the Ethernet chip set and forwarded to the FPGA. The FPGA uses the Ethernet and IVDS header information of these packets to determine what needs to be done with the display data within the packets. The FPGA then stores the incoming display data in the appropriate memory unit.

The FPGA is also responsible for updating the audio DAC at a 22.1 KHz rate. The 22.1 KHz frequency is necessary to support audio code transmission. Furthermore, the FPGA must copy pixels from the Buffer Memory into the Main Memory as well as performing read operations from the Main Memory every 74.074 ns to update the NTSC encoder as discussed in Appendix A.

The internal FPGA design can be divided into four subsystems. The packet processing subsystem is responsible for receiving the Ethernet packets from the Ethernet chip set. The Buffer Memory management subsystem provides for the addressing and data input to the Buffer Memory. The Audio Memory management subsystem performs the addressing as well as reading and writing to and from the Audio Memory. The Main Memory management subsystem is responsible for maintaining the visual display by continuously updating the NTSC Encoder with 24-bit RGB pixels from the Main Memory. The Main Memory management subsystem also maintains a comparator that determines when a pixel is copied from the Buffer Memory to the Main Memory.

The following internal FPGA block descriptions are the first version of this design. FPGA design issues presented in the next chapter forced slight changes to be made to the internal subsystems. These changes reflect the final design of this FPGA and will be presented in the next chapter as well.

6.2 Packet Processing Subsystem

6.2.1 Shift Registers

The Ethernet chip set uses the seven-byte Preamble field of the Ethernet packet for synchronization and strips that field off before forwarding the packet to the FGPA. Ethernet packet data is clocked across four parallel lines from the Ethernet chip set at a rate of 25 Mbps. Each of the four data lines is fed into a series of parallel 16-bit shift registers inside the FPGA as shown in Figure 6.1. The 25 MHz clock supplied by the Ethernet chip set is used to clock the data through the shift register.

The series of 16-bit Shift Registers are designed to simultaneously hold 192 bits of data. The 192 bits of data make up all of the Ethernet and IVDS header data as well as three bytes of the IVDS packet data field. Control logic is always looking at the bits as they are shifted through the Shift Registers.

6.2.2 Delimiter and Address Verification

When the head of a packet first enters the FPGA and exactly fills the Shift Registers, control logic will verify that the Ethernet packet has a valid Delimiter field and that the Destination Address field contains the address of this Decoder.

Each Decoder has a unique address that is set using DIP switches on each Decoder board. Each Decoder board address is ten bits, allowing the IVDS Browser to serve 1024 Decoder Units.

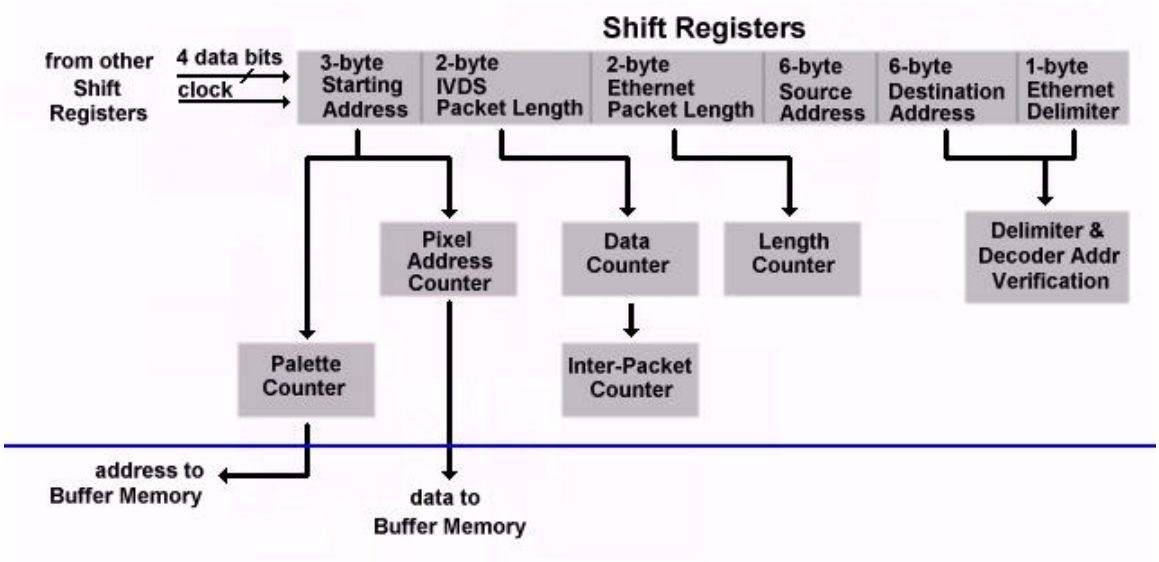


Figure 6.1: Packet Processing Subsystem

In order for the control logic to recognize the head of an Ethernet packet, the control logic will look at the received bits each clock cycle as they are shifted through the Shift Registers.

Control logic will verify the header of an Ethernet packet by using a series of inverters and AND gates to detect the Ethernet Start Frame Delimiter as seen in Sheet 1 of the FPGA schematic in Appendix B The value of the DIP switches will be compared against the first ten bits in the Ethernet Destination Address field. If the received packet is not addressed to this Decoder, the FPGA will ignore this packet but continue to shift it through the Shift Registers.

The result of the address comparison will be ORed with the eleventh bit in the Destination Address field. If the eleventh bit is set to a logic high, the Browser is in broadcast mode and intends for every Decoder to accept the incoming Ethernet packet.

The Delimiter verification and the address verification will be ANDed together to verify that the beginning of an Ethernet frame is detected and the packet is intended for this Decoder. It should be noted that the bits in the Source Address field are not used by the FPGA.

6.2.3 Length Counter

When a new Ethernet packet is verified, the value in the Ethernet Packet Length field is loaded into the Length Counter implemented within the FPGA. The Length counter is an 11-bit loadable decrementing counter. The clock to this counter is provided by the 25 MHz clock from the Ethernet chip set. Each 25 MHz clock cycle represents four bits of received data. A flip-flip is used to increment the Length Counter every second tick of the 25 MHz clock so that the counter counts by bytes rather than 4-bit nibbles.

Every bit of the counter's value is ORed together resulting in a value that is only zero when the Length Counter is zero. The Length Counter only decrements to zero when the end of the Ethernet packet has been reached.

The ORed Length Counter value will be inverted and ANDed with the Delimiter and address verification results. The resulting signal will only transition to a logic high when a new Ethernet packet is detected and there is no current Ethernet packet being

received. This ensures that a false header and destination address is not detected due to random bits within the Ethernet packet.

6.2.4 Data Counter

The ORed Length Counter value also acts as an enable for the Data Counter. The Data Counter is a loadable 11-bit counter implemented within the FPGA. The halfed 25 MHz clock from the Ethernet chip set is used to decrement this counter. The Data Counter is loaded from the shift register locations that hold the IVDS Packet Length. This counter will be loaded when a new Ethernet packet is detected or when a new IVDS packet is detected. A new IVDS packet will be detected when multiple IVDS packets are sent within a single Ethernet packet.

Each IVDS packet has a length field that is loaded into the Data Counter. The Data Counter is used to determine when the current IVDS packet has ended. The bits that make up the value of this counter are ORed together resulting in a signal that will remain in a logic high state as long as there is a valid IVDS packet being received.

The ORed Data Counter value serves as an enabling signal to many of the counters in the Buffer Memory management subsystem of the FPGA. When an IVDS packet is being received, the Data Counter will enable the counters in the Buffer Memory management subsystem to function and allow processing the incoming display data.

When the IVDS packet ends, the ORed Data Counter value will transition to a logic low, disable some of the Buffer Memory management subsystem counters, and enable the Inter-Packet Counter to begin decrementing.

6.2.5 Inter-Packet Counter

When an IVDS packet ends, the Ethernet packet may also end or a new IVDS packet may immeadiately follow within the same Ethernet packet. If the Ethernet packet ends, the Length Counter will have reached zero and will disable the Data Counter as well as the Inter-Packet Counter.

When the end of an IVDS packet is encountered and the Ethernet Length Counter value has not reached zero, the Inter-Packet Counter is allowed to begin decrementing from a preloaded value. There are a fixed number of clock cycles required to shift the next IVDS packet header through the Shift Registers into alignment with the counters into which they are to be loaded. When Inter-Packet Counter value decrements to zero, this alignment occurs. The Inter-Packet Counter's zero value generates a load signal which allows processing of the new IVDS packet to occur.

This load signal forces the Inter-Packet Counter to stop decrementing are load the value from which it begins decrementing from when activated. The load signal causes the IVDS packet corresponding fields to be loaded into the Data Counter, Pixel Address Counter, Palette Counter, and Mode Register. The first three bytes of display data are also at the position in the Shift Registers where they can be loaded into the Data Latches to be processed. The remaining display data bits of the new IVDS packet are shifted into the FPGA and processed accordingly.

The packet processing subsystem provides the FPGA with the ability to look at both the Ethernet and IVDS header data simultaneously. Counters are implemented to

identify when the Ethernet and IVDS each end, as well as provide the ability to wait while a new IVDS packet is loaded into the Shift Registers.

6.3 Buffer Memory Subsystem

The Buffer Memory Subsystem, seen in Figure 6.2, contains the logic blocks that both address and provide data to the Buffer Memory. Logic within this subsystem also determines what mode the FPGA should operate in so as to process the display data correctly. The RGB representations of the pixels are buffered during their conversion from 8-bit to 24-bit in this subsystem.

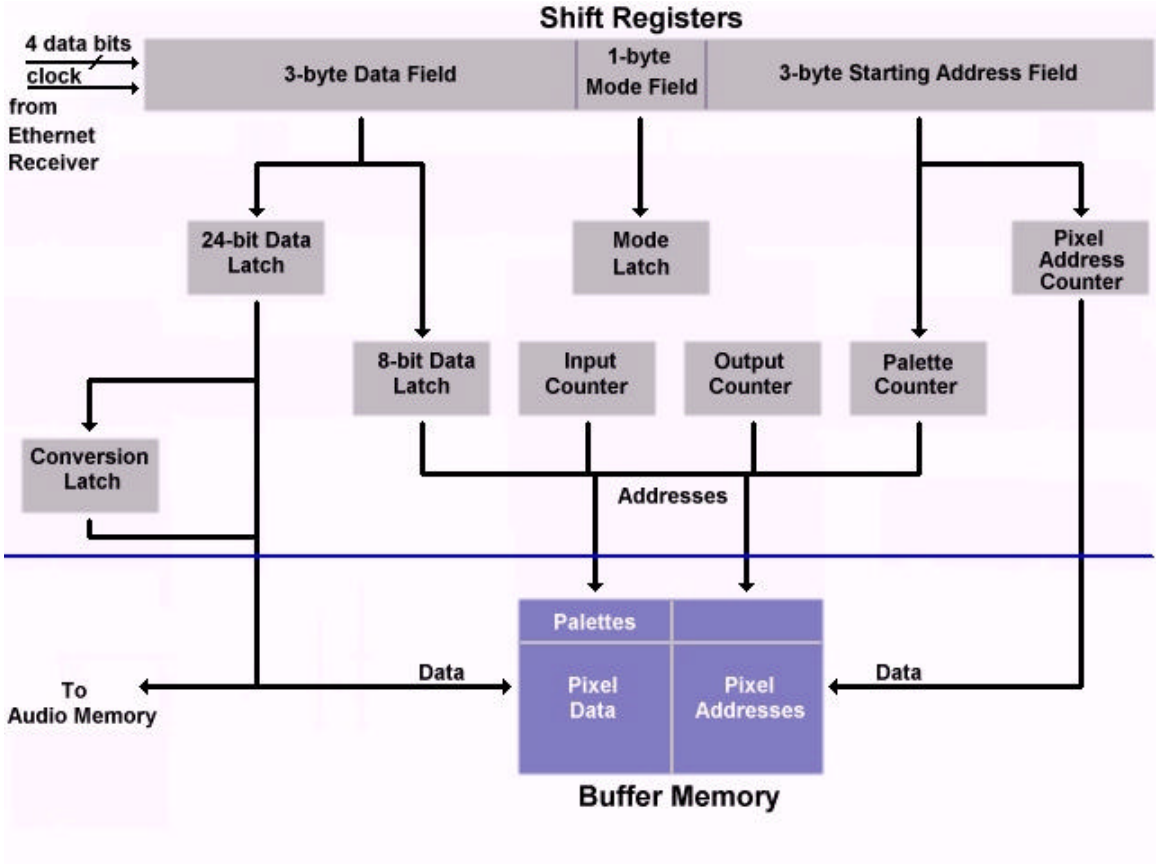


Figure 6.2: Buffer Memory Subsystem

6.3.1 Mode Latch

The 8-bit Mode Latch and its surrounding control logic determine what mode the FPGA will process the display data in. The Mode Latch latches its data from the shift register outputs corresponding to the mode field of the IVDS packet header. The enable latch signal is generated by both the Ethernet control logic and the Inter-Packet Counter eact time a new IVDS packet arrives.

Once the mode bits are latched and fed through a block of control logic to determine te mode, every display data unit will be processed identically for the duration of the IVDS packet. To change modes for a different type of display data, the Browser must send a new packet with the appropriate mode field settings.

6.3.2 24-Bit Data Latch

The 24-Bit Data Latch buffers the display data units received from the data field of an IVDS packet. The output of the 24-Bit Data Latch is used as data for the Buffer Memory and Audio Memory. For the duration of that IVDS packet, the frequency in which the 24-bits of IVDS data field are latched from the Shift Registers' outputs remains constant.

The 24-Bit Data Latch output is tri-stated onto two different buses. In the audio mode, 16 of the 24 bits are tri-stated onto the Audio Bus for storage in the Audio Memory. In the 24-bit pixel mode and the load palette mode, the 24 bits are tri-stated onto the Pixel Data Bus.

In the 24-bit pixel mode, these 24-bits represent a single pixel of 24-bit RGB color. In the load palette mode, these 24-bits represent one 24-bit palette entry. In the 8-bit pixel mode, these 24-bits represent three 8-bit pixels. Because additional logic would be required to disable the latch enable signal in the 8-bit pixel mode, the 24-Bit Data Latch is allowed to latch in the data. In the 8-bit pixel mode this latch is loaded, however its tri-state output is never enabled. In these three modes, a new 24-bit display data unit is latched every 240 ns.

In the audio mode, these 24-bits represent one 16-bit sample. A new 24-bit display data unit is latched every 160 ns. Sixteen of these 24 bits form one audio sample. The other eight bits are part of the next audio sample. Because audio samples are stored in 16-bit increments, only 16 bits from the 24-Bit Data Latch are placed on the Audio Bus.

In another 160 ns, the next audio sample, 8 bits of which were latched last cycle, will have shifted into the correct position. Then this new sample and part of the next will be latched into the 24-Bit Data Latch.

The 24-Bit Data Latch is the interface through which the display data is extracted from the data field of the IVDS packet and then stored in its respective memory units. In the 24-bit pixel mode, 24-bit pixels are stored in lower 24-bit of the 48-bit slices that make up the Buffer Memory. In the upper 24-bits of each 48-bit slice, the address that corresponds to each pixel must be stored with it in the Buffer Memory. The Input Counter provides the address in which to store each 24-bit pixel in the Buffer Memory.

6.3.3 Pixel Address Counter

When the FPGA is operating in 8-bit or 24-bit pixel mode, the Pixel Address Counter output serves as data to the Buffer Memory. The Pixel Address Counter is a 24-bit, loadable, incrementing counter, whose initial value is loaded from the Starting Address field of the IVDS packet. The load signal is triggered by the validation of a new Ethernet packet or the arrival of a new IVDS packet as signaled by the Inter-Packet Counter. The clock is provided by the write signal generated to write data into the Buffer Memory.

The value that is loaded when each IVDS packet first arrives, is the 24-bit address in the Main Memory where the first 24-bit pixel should be stored. Before each 24-bit pixel can be stored in Main Memory, it must be temporarily stored in the Buffer Memory to wait for the proper time to be copied into the Main Memory.

This counter increments its count for each subsequent 24-bit pixel arrivals for the remainder of the IVDS packet. Thus, the Browser only needs to send the address corresponding to the first pixel in the IVDS data field. The Pixel Address Counter will properly address all other pixels in an IVDS packet.

6.3.4 8-Bit Data Latch

The 8-Bit Data Latch is a series of three 8-bit latches with tri-state outputs. When the Decoder is in the 8-bit pixel mode, each latch is loaded from the data field of the IVDS packet. Each latch is loaded every 240 ns with eight bits from the shift register outputs corresponding to the IVDS packet data field.

When 8-bit pixels are received by the Decoder, they must be converted into 24-bits before they can be stored in the Buffer Memory. Each pixel is converted using color palette lookup tables stored in the Buffer Memory.

When a packet is received containing 8-bit pixels, four bits in the mode field specify which of the 16 possible palettes to use for conversion. In order perform this conversion, the 8-bit pixels are used in conjunction with these four bits latched into the Mode Latch to address the Buffer Memory.

The palettes are stored in the first 4K of the Buffer Memory such that the 8-bit pixel colors can be used within each palette and the four palette bits are used to select the palette to use. Each palette has 256 entries and the 8 pixel bits address those 256 entries. The four palette bits address the appropriate 256 entry palette to use as shown in Figure 6.3.

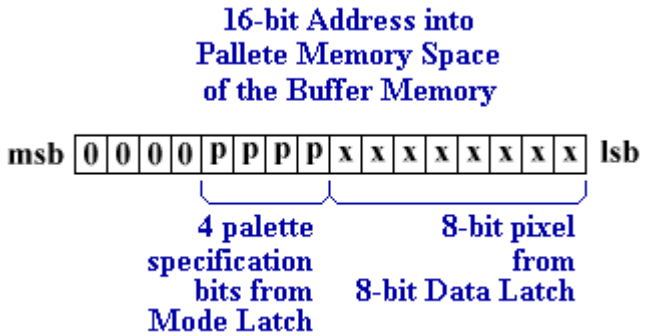


Figure 6.3: Four Palette Bits and 8 Pixel Bits Form Address

A read operation is used to obtain the 24-bit pixel color corresponding each received 8-bit color pixel. The combined 8-bit pixel color and 4-bit palette number addresses the Buffer Memory and the 24-bit color pixel entry at that address is read out. The 24-bit color pixel is buffered in the Conversion Latch for one clock cycle and stored

back in the Buffer Memory with it corresponding address provided by the Pixel Address Counter.

Each of the three 8-bit color pixels will be used separately to address the Buffer Memory. The corresponding 24-bit pixel will be buffered and stored into the Buffer Memory to be available for copying into the Main Memory. The control logic used to coordinate these functions will be discussed later in the FPGA Development chapter.

6.3.5 Conversion Latch

The Conversion Latch is a 24-bit, tri-state output latch. This latch is used in the 8-bit pixel mode to buffer each 24-bit pixel colors read from the Buffer Memory. The value read out on the data lines from the Buffer Memory is the 24-bit representation of the received 8-bit pixel.

The control logic activates the latch enable signal as part of this read cycle. The 24-bit pixel read from the Buffer Memory is then latched into the Conversion Latch. During the next clock cycle, the latch output enables are activated and a write cycle to the Buffer Memory is initiated by the control logic. During this write cycle, the 24-bit pixel color is stored in the lower half of a 48-bit slice of the Buffer Memory in the same way as a normal 24-bit pixel. The corresponding screen address of this pixel is the Pixel Address Counter value stored in the upper 24-bits of that same slice. The Input Counter provides the address in which to store each 48-bit slice in the Buffer Memory during write cycles.

6.3.6 Input Counter

The Input Counter is a loadable, 16-bit counter with tri-state buffers on the counter outputs. This counter is used to address the Buffer Memory during the write cycles in which the 24-bit pixels are stored with their corresponding screen addresses in 48-bit slices. The tri-state buffer outputs are enabled only during write cycles to the Buffer Memory. These write cycles only occur in the 8-bit pixel and 24-bit pixel modes. The Input Counter is then incremented at the end of the write cycle.

When the Input Counter reaches its highest possible value, additional logic is triggered causing the Input Counter to load the first address after the palette storage. This is necessary because if the counter were allowed to reset naturally, it would address the palettes storage area in the Buffer Memory. Thus, additional logic is needed so that the Input Counter can only provide addresses within the pixel storage memory space. The memory space allocation within the Buffer Memory is shown in Figure 5.5 found in Chapter 5.

6.3.7 Output Counter

The Output Counter is a loadable, 16-bit counter with tri-state buffers on the counter's outputs. This counter is used to address the Buffer Memory during read cycles in which the 24-bit pixels are copied into the Update Latch and the corresponding pixel address value is placed in the Pixel Address Comparator. The Update Latch and Pixel Address Comparator are described in the Main Memory Subsystem section of this chapter.

In order to simplify control logic, the read operation from the Buffer Memory occurs every cycle that is reserved for it, independent of whether a match occurs in the Pixel Address Comparator. This does not cause a problem because the Output Counter is not allowed to increment unless a match is made in the Pixel Address Comparator. So the same 48-bit slice will be repeatedly read from the Buffer Memory until a match is made and the 24-bit pixel is copied into the Main Memory. Only then will the Output Counter be allowed to increment and the next pixel be read and latched into the Update Latch.

The control logic must be designed so that this read operation can occur once every 74.074 ns. This will allow pixels to be copied from the Buffer Memory into the Main Memory at the same rate that the Main Memory Subsystem updates the NTSC Encoder. Furthermore, this functionality must be designed to occur independent of the mode that the Decoder is currently in.

When the NTSC Address Counter arrives at a memory address that matches the address in the comparator, the pixel held by the Update Latch can be written into the Main Memory and used to update the NTSC encoder. At this point, a new pixel and corresponding address needs to be available in the Update Latch and Comparator within 74.074 ns.

Making a pixel-address pair available in the Update Latch and Comparator every 74.074 ns allows the Main Memory Subsystem and Buffer Memory Subsystem to synchronize and copy the pixels into the Main Memory at the same rate in which the NTSC encoder is updated. These systems will remain synchronized as long as each pixel's associated screen address is consecutive.

If these subsystems could not synchronize, one pixel would be copied into the Main Memory and the NTSC Address Counter would increment to the next address before the new pixel intended for that address would be available in the Update Latch. Then each pixel would have to wait 33.3 ms until the NTSC Address Counter cycled through the entire Main Memory and reached the matching address. That would create severe delays in the overall time required to update the entire NTSC display.

Similar to the Input Counter, the Output Counter is a loadable counter. When the counter reaches its maximum value, the first address of the pixel memory space within the Buffer Memory is loaded in the next clock cycle. This prevents the Output Counter from addressing contents of the Buffer Memory outside the pixel storage memory space in forwarding corrupt pixel data to the Main Memory Subsystem.

6.3.8 Palette Counter

The Palette Counter is a loadable, 12-bit counter with tri-state buffers on the counter's outputs. When a new IVDS packet arrives, the Palette Counter is loaded with the first twelve bits from the Data Length field of the IVDS packet regardless of the mode. The 12-bit Palette Counter is only capable of addressing locations within the palette memory space of the Buffer Memory.

While the Palette Counter is loaded regardless of the mode, it is only used in the load palette mode. In the load palette mode, the Palette Counter is used to address the Buffer Memory when each new 24-bit palette entry is written. In all other modes, the

Palette Counter is not allowed to increment and it's tri-state output buffers will not be enabled.

6.4 Audio Memory Subsystem

The Audio Memory subsystem, shown in Figure 6.4, stores incoming audio samples into the Audio Memory as well as reads audio samples from the Audio Memory. Two counters, the Input Audio Counter and the Output Audio Counter, are implemented in the FPGA to address the Audio Memory for the writing and reading operations. Additionally, a comparator is implemented to prevent the Output Audio Counter from counting past the value of the Input Audio Counter.

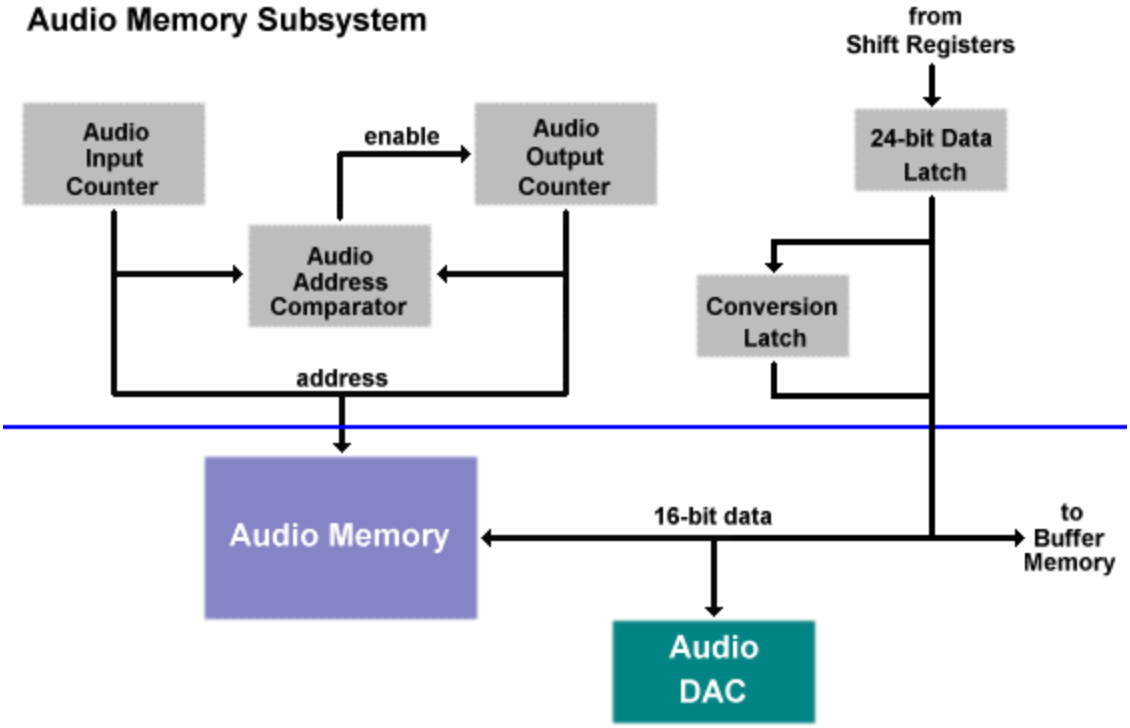


Figure 6.4: Audio Memory Subsystem

6.4.1 24-Bit Data Latch

This latch was initially discussed in the Buffer Memory Subsystem section. The latch enable signal is triggered to latch in 24-bit segments of the IVDS data field from the Shift Registers regardless of the mode. However, in the audio mode, control logic latches 24-bit segments of data every 160 ns.

Each 24-bit segment latched consists of one 16-bit audio sample and half of the next audio sample. 160 ns later the next audio sample will have shifted into position so that it can be latched correctly in the 24-bit Data Latch with half of the next audio sample.

The tri-state output buffers allow data to be output from this latch in 16-bit increments rather than the 24-bit increments used in the 24-bit pixel mode. These 16-bit increments are written into the Audio Memory. The other 8-bits are part of the next audio sample.

6.4.2 Audio Input Counter

The Audio Input Counter is 20-bit, incrementing counter. It is used to address the Audio Memory when an audio sample is written into memory. When this counter reaches its maximum value it will reset and begin counting again. This counter is allowed to increment normally when the FPGA receives audio samples in continuous audio mode. If an IVDS packet is received that sets the FPGA in new audio mode, this counter is reset and begins to increment from zero.

If the Browser sends more audio data than the Audio Memory can store, the Audio Memory Subsystem will fill the Audio Memory, reset, and begin overwriting data

already stored. The Browser is responsible for not transmitting more audio data than the Audio Memory can hold before it is read and latch by the board-level shift registers.

6.4.3 Audio Output Counter

The Audio Input Counter is 20-bit, incrementing counter. It is used to address the Audio Memory when an audio sample is read from memory. When this counter reaches its maximum value it will reset and begin counting again. This counter is allowed to increment normally when the FPGA receives audio samples in continuous audio mode. If an IVDS packet is received that set the FPGA in new audio mode, this counter is reset and begins to increment from zero.

When an audio sample is read from the Audio Memory, a pair of chips on the Decoder board containing 8-bit shift registers latch the audio sample. A clock signal, provided by control logic in the FPGA allows the shift registers shift the audio sample bits one by one into the Audio DAC. The Audio DAC receives these bits on it one input data line. Inside the Audio DAC the received data is shifted into an internal shift register. The received audio sample is then converted into analog and embedded into the NTSC signal.

The Audio Input Counter uses a 22.1KHz clock to read an audio sample from memory every 45,249 ns. Audio samples are received and stored at a much faster rate than they are read out and latched by the board-level shift registers. The board-level shift registers will only latch audio samples during read cycles.

6.4.4 Audio Address Comparator

The Audio Address Comparator is a 20-bit comparator. It is used to compare the current count values held by the Audio Input Counter and the Audio Output Counter. Suppose both of these counters are reset to zero and a series of audio samples are received and stored. The Audio Output Counter will be used to read these samples out to the Audio DAC. The Audio Address Comparator is used to prevent the Audio Output Counter from counting past the Input Audio Counter.

The Audio Output Counter is allowed to have a higher values than the Audio Input Counter. However, it is not allowed to cycle through the Audio Memory and increment its value equal to and then higher than the Input Audio Counter.

When this situation occurs and the Audio Address Comparator finds the counter values equal, it will disable all read cycles and not allow the Output Audio Counter to increment until new IVDS packet audio samples are received and stored in memory. This will prevent the Output Audio Counter from reading areas of the Audio Memory that contain audio samples that have already been played or contain no audio samples at all.

6.5 Main Memory Subsystem

The Main Memory Subsystem, shown in Figure 6.5, has two responsibilities. A read operation from the Main Memory must be performed every 74.074 ns. The NTSC Encoder latches the 24-bit pixel data read from the Main Memory and converts it to an analog forms which is input to the Composite Video Generator.

In its second responsibility, the Main Memory Subsystem uses a comparator to detect that a 24-bit buffered pixel should to be copied into the Main Memory. In this situation, rather than performing the usual read from the Main Memory to update the NTSC Encoder, a write is performed using this new 24-bit pixel. This write operation copies the 24-bit pixel from the Update Latch into the Main Memory as well as updating the NTSC Encoder with this new pixel.

6.5.1 NTSC Address Counter

The NTSC Address Counter is a 20-bit, incrementing counter. The value held by this counter is used to address the Main Memory for both read and write operations. Each clock cycle, the counter's value also serves as an input to the Pixel Address Comparator.

307,200 24-bit pixels are required for the 640x480 display that the user will see. The 512Kx24 Main Memory however, is capable of storing 512,000 24-bit pixels. The Main Memory Subsystem will perform reads to update the NTSC Encoder regardless of the status of any of the other subsystems within the Decoder.

Some of the pixels read from the Main Memory will be unused because they will be read during the horizontal or vertical retrace periods of the display generated by the NTSC Encoder. Pixels read during these times will be latched by the NTSC Encoder, but the NTSC Encoder will not use these pixels.

Therefore, extra pixels are needed in the locations of the Main Memory that will be read during these retrace periods. The Main Memory will not utilize all of the 512,000 memory addresses. At some point before the last location of the Main Memory, the last

pixel required to update the screen will be reached. This pixel value however, will not be used by the NTSC Encoder because a vertical retrace period will occur when the NTSC Encoder reaches the bottom of the display.

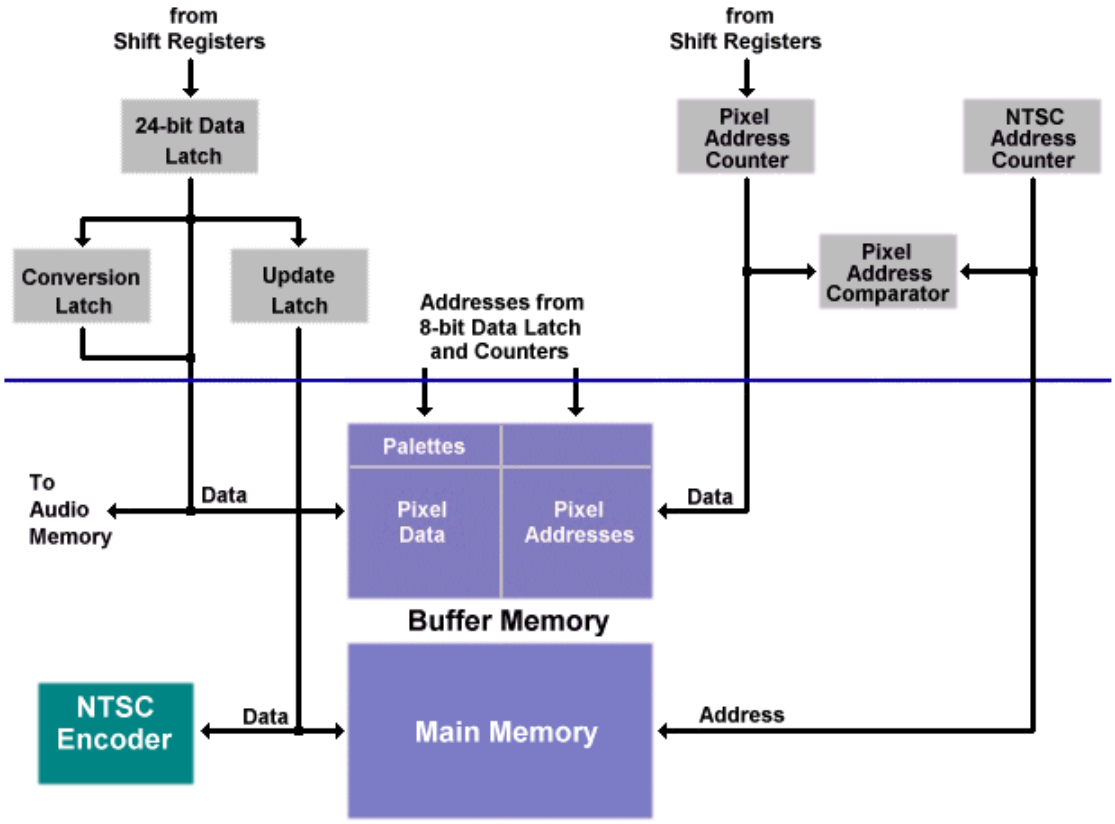


Figure 6.5: Main Memory Subsystem

After the vertical retrace, the next pixel value that the NTSC Encoder expects is the first pixel at the top of the display. Thus, after the last pixel of the vertical retrace, the NTSC Address Counter will have to be reset. The value that will cause the NTSC Address Counter to reset is going to be determined at a later time during testing.

Futhermore, the actual television image is generated by illuminating all even number lines on the television screen, performing a vertical retrace, and then illuminating all odd number lines. The Browser software would be responsible for forwarding pixels to the Decoders according to this illumination scheme.

6.5.2 Pixel Address Comparator and Update Latch

The Pixel Address Comparator is a 20-bit comparator with a 20-bit latch feeding one of the two pair of inputs. The latched input is fed by the upper 24-bits of the 48-bit slices read from the Buffer Memory. These upper 24-bits represent the address in the Main Memory where the pixel value stored in this slice is to be stored. The non-latched input is fed by the NTSC Address Counter.

The Update Latch is a 24-bit latch with tri-state output buffers. The inputs to this latch come from the lower 24-bits of the 48-bit slices read from the Buffer Memory. These lower 24-pixels represent the 24-bit RGB pixel color to be stored in the Main Memory. The enable signal of the output buffer is controlled by the Pixel Address Comparator.

From the Buffer Memory, a 24-bit pixel and the corresponding 24-bit address where it is to be stored in the Main Memory are read in a single 48-bit slice. The 24-bit pixel is latched into the Update Latch during this read operation. The least significant 20 bits of the 24-bit address is latched into the Pixel Address Comparator. Only 20 of the 24 bits are required to address the Main Memory.

Meanwhile, the NTSC Address Counter is cycling through its possible values. Each clock cycle the counter value is input into the Pixel Address Comparator where it is compared with the address latched from the Buffer Memory read operation. If the addresses do not match, the usual read operation is performed on the Main Memory updating the NTSC Encoder with the next 24-bit pixel value.

If the addresses do match, then the Update Latch holds the new value that should be stored in the Main Memory at the location to be addressed by the NTSC Address Counter. The Update Data Latch's tri-state output buffers are then enabled and a write operation is performed using the 24-bit pixel released from the Update Latch. The new 24-bit pixel is stored in the Main Memory and is latched by the NTSC Encoder to be used to refresh the NTSC display.

Additionally, when the addresses match, the Output Counter is allowed to increment. Then, in the next read cycle of the Buffer Memory, the next 24-bit pixel and its corresponding address will be latched into the Update Latch and Pixel Address Comparator.

If the next NTSC Address Counter value matches the next address now held in the Pixel Address Comparator another write cycle is initiated. The next pixel is written into the Main Memory, used by the NTSC Encoder, and the Output Counter increments. This cycle continues every time the address match occurs.

Otherwise, read cycles are performed as scheduled on the Buffer Memory using the constant value held by the Ouput Counter. Each read cycle, the same pixel value is latched into the Update Latch and the same corresponding address is latched into the Pixel

Address Comparator. The same value will be continually read until a match is made within the Pixel Address Comparator. Then the Output Counter will be incremented and a new pixel and address will be read. The allowance of these repeated read operations from the Buffer Memory rather than only reading when necessary simplifies the control logic within the FPGA.

In the worst case, the pixel remains latched in the Update Latch for 33.3 ms, until the NTSC Address Counter cycles through every possible Main Memory address. At that point, a match will be made and the pixel written into the Main Memory.

6.6 FPGA Internal Control Logic Clock Requirements

In 8-bit pixel mode a triplet of 8-bit pixels will be loaded in the 8-Bit Pixel Latch every 240 ns. Therefore each 8-bit pixel must be converted and stored in the Buffer Memory in 80 ns. As discussed eariler in this chapter, an 8-bit pixel is used in a read operation to address the palette storage memory space within the Buffer Memory. The 24-bit pixel read out is latched into the Conversion Latch. That 24-bit pixel value is then written back into the Buffer Memory with its associated 24-bit Main Memory address.

The above operation requires two cycles, however a third cycle is required for two reasons. First, a small amount of time must pass in order for the data to be latched in the Conversion Latch and made available on the output lines. This middle cycle would occur between the read and write operations on the Buffer Memory. Additionally, this middle cycle is required so that read operations for potential updates of the Main Memory can occur.

In order to obtain three operational cycles in 80 ns a clock period of 26.67 ns is required. Therefore, an additional clock would be required to provide the synchronization for the above cycles in the FPGA. This second clock would come from an external clock on the Decoder Board. In an effort to minimize the number of board level components, this clock will be a multiple of the 13.5 MHz clock for the NTSC Encoder.

This external clock would provide a 81 MHz frequency. The clock would be halved to obtain a 40.5 MHz signal for the control logic within the FPGA. Additionally, this clock would be divided by six to obtain the 13.5 MHz clock for the NTSC encoder. The 40.5 MHz signal would provide 24.69 ns clock cycles within the FPGA, thus providing the three operational cycles within 80 ns.

6.7 Conclusion

This chapter has introduced and provided a block level description of the the four subsystems that make up the internal design within the IVDS Decoder's FPGA. In the course of these descriptions it was made clear how the display data flows through each of these subsystems and how they interact.

The next chapter will discuss the issues encountered that forced a slight modification to this internal FPGA design and provide a description of the four subsystems after this modification.

Chapter 7. Modification of the Block Level Design of the FPGA

7.1 Introduction

This chapter will present several clock and delay issues encountered that forced modifications to the logic design to be placed within the FPGA. The effects of the design modifications will be presented at the same functional component level as presented in Chapter 6.

7.2 Delimiter and Address Verification Logic Delay and Design Modification

The delay through the logic that detects and verifies the delimiter and device address of the Decoder must be less then a 40 ns clock cycle provided by the Ethernet chip set. In order to meet this requirement, the number of bits in the device address had to be reduced from ten to eight. This reduced the number of levels of AND gates and the associated delay required to compare all of the bits and generate an enable signal to the logic that must load the bits from the Shift Registers. Assuming 6 ns per logic level, the new delay to verify the Ethernet delimiter and device address was below 40 ns.

7.3 FPGA Interface Logic Delay and Design Modification

7.3.1 FPGA Interface Logic Delay

Some initial compiling of the logic design revealed that while the internal FPGA circuitry appeared to be able to function at the desired 40.5 MHz clock rate, the delay introduced by the interface logic to the pins of the FPGA itself was 19.7 ns. So the 54.4 ns delay through the FPGA pin interface logic, through the 15 ns SRAM, and through the

interface logic back onto the FPGA would exceed the allowable 24.69 ns for a cycle. Therefore, FPGA could not be made to work in the 8-bit pixel mode.

In the 24-bit pixel and palette mode, only one write operation to the Buffer Memory needs to occur every 240 ns. In both audio modes, a write must be performed to the Buffer Memory every 160 ns. In the audio, palette, and 24-bit pixel modes, a read operation must be performed at least once every 74.074 ns to copy pixels in the Main Memory. Both read and write operations would require 34.7 ns through the FPGA pin logic and the 15ns SRAM. It appeared that the remaining modes could be made to work within the delay limits.

7.3.2 Block Level Modification of FPGA Design

In order to solve the 8-bit pixel mode delay problem, it was decided that 8-bit pixels received by the FPGA would be stored in their 8-bit form in the usual 24-bit field of the 48-bit Buffer Memory slices. An 8-bit pixel will be converted into a 24-bit value when the read operation is performed normally to obtain a pixel to store in the Update Latch.

Usually, 24-bit pixels are read from the Buffer Memory and latched into the Update Latch. However, 8-bit pixels read from the Buffer Memory will be input to three Am27S23 256x8 PROMs that together, will convert the 8-bit color into a 24-bit color. It should be noted that while the initial design provided 16 different palettes to convert pixel values, the PROMs provide only one palette to use.

These three PROMs also contain internal latches with tri-state outputs that control each PROM's output signal. The resulting value read from each of these PROMs will be

buffered in each PROMs internal latch. The PROMs enable their tri-state outputs placing the new 24-bit pixel value onto the Main Memory bus according to the same control signals as the Update Latch. Thus, in the 8-bit pixel mode these PROMs convert the 8-bit pixel into a 24-bit pixel and then serve the purpose as the Update Latch does for the 24-bit pixel mode.

7.3.3 Modified FPGA Internal Logic Design

This modification allows several blocks of logic presented in Chapter 6 to be removed because they will no longer serve any purpose in this modified design. Because the palette conversion function now resides in the PROMs, the Palette Counter, the 8-Bit Pixel Latch, and the Conversion Latch can be removed from the design. Furthermore, the four palette bits in the mode field of the IVDS packet will no longer be used. The specification for the remaining four bits in the mode field remains unchanged.

A method of distinguishing between 8-bit pixels and 24-bit pixels stored in the Buffer Memory is required. Depending upon which type of pixel value is stored, either the Update Latch or the PROMs must be used.

24-bits are reserved in the upper half of each Buffer Memory slice for the Main Memory address to reside. The Main Memory address however, is only 19-bits in length. In the initial design, the remaining six bits would have been filled with zeros. Now however, when each pixel value is stored in its 48-bit slice, a bit from the unused six bits in the upper 24-bit piece of the memory slice will be set according to the length of the pixel value being stored. If a 24-bit pixel value is being stored, this pixel type flag bit will be a

logic low. If an 8-bit pixel value is being stored, this pixel type flag bit will be a logic high. This flag will be set according to the mode received in the IVDS packet.

When each pixel value and associated Main Memory address are read from the Buffer Memory, 8-bits of the pixel value will be sent to the PROMs and all 24-bit reads will be latched into the Update Latch. The pixel type flag will determine whether the output enables of the Update Latch or the PROMs are enabled if the Pixel Address Comparator signals a match.

7.3.4 New Timing Requirements of IVDS Modes

This modification decreases the speed in which read and write operations must occur to the Buffer Memory. In 8-bit pixel mode, each pixel must be written into the Buffer Memory within 80 ns. Additionally, a cycle must be reserved for a read operation in this 80 ns period that will buffer a pixel in the Update Latch or the PROMs.

The 19.7 ns delay through the FPGA interface logic, in addition to the 15 ns Buffer Memory SRAM delay, will be small enough to allow a read or write operation to be performed every 40 ns. The 24-bit pixel mode reading and writing timing requirements were presented above and remain the same after this modification. The palette mode that previously stored palette entries in the Buffer Memory is no longer needed and removed from the design. Finally, these modifications do not effect the FPGA function in the audio modes.

7.4 Conclusion

This chapter presented two modifications in the FPGA based logic design for the IVDS Decoder. The delay through the logic to verify the Ethernet delimiter and device address caused the number of bits in the device address to be reduced. The FPGA interface delay to its pins resulting in a modification of the FPGA logic system concept. As a result of this modification, three board level PROMs were added and several logic blocks in the FPGA based design were removed. The next chapter will present the current status of the FPGA based logic design.

Chapter 8. Present FPGA Design Status

8.1 Introduction

This chapter will present the current status of the logic design for the IVDS Decoder FPGA and use timing diagrams to show the current functionality. Funding for this project ran out before this design could be completly implemented. Therefore, certain portions of the logic design have been completed and simulated, while other portions of the logic design are incomplete.

8.2 Input Assumptions

The Ethernet chip set interfaces to the FPGA using the MII. Documentation provided with the DP83840 Ethernet Physical Layer chip was not clear on the exact timing relationships between the signals. The signal paths between the Ethernet chip and the FPGA were made at the board level for all of the signals involved in the receiving function. A basic set of assumptions was made regarding this interface.

The logic design and simulation began using only RX_CLK, the 25 MHz clock signal, and RXD[3:0], the four data lines from the Ethernet chip. The data was clocked into the FPGA on the rising edge of the clock signal. I have assumed that the 25 MHz clock is only forwarded to the FPGA when data is being sent as well.

The remaining MII signals are the CRS, RX_DV, RX_ER signal descriptions. The CRS (Carrier Sense) pin is asserted high to indicate the presence of carrier due to receive or transmit activities [22]. The RX_DV or DV (Receive Data Valid) signal is asserted high to indicate that valid data is present on RXD[3:0] [22]. The RX_ER or ER (Receive

Error) signal is asserted high to indicate that an invalid symbol has been detected inside a received packet in the 100 Mbps mode [22]. The exact timing of activities on these three signals would be determined when a prototype unit was manufactured and testing could begin.

The control logic will use the 40.5 MHz clock signal, INT_CLK, to generate the control signals for read and write operations involving the Buffer Memory. The NTSC Memory Subsystem will use the 13.5 MHz clock input to the FPGA by the board level NTSC Encoder.

8.3 FPGA Design Status

8.3.1 Shift Registers

The shift registers have been implemented and tested. Each of the four data signals from the MII are shifted in parallel through three 16-bit shift registers. In the final design, the shift register reset signal will need to be grounded. The clock enable signal of the first two groups of shift registers on Sheets 2 and 3 of the schematic in Appendix B will need to be tied to a logic high. The clock enable signal for the group of shift registers on Sheet 1, will be controlled by an unfinished block of control logic seen in the lower left corner of Sheet 1.

This control logic will hold the CE signal high, allowing bits to shift to the Ethernet delimiter and address verification control logic when the FPGA is not processing an Ethernet packet. When an Ethernet packet is detected, the unfinished control logic will force this CE signal low while the Length Counter is decrementing. When the Length

Counter reaches zero, the end of the Ethernet packet has effectively been reached. The control logic changes the CE signal to a logic high, allowing the 4-bit nibbles of the next packet to shift through this group of shift registers.

During the design of this control logic, the interface to the Ethernet chip set was not clear. A better approach to this control logic would be a counter that would count as the nibbles were shifted through the shift registers. The counter would be clocked using the clock from the MII. When the counter reaches a certain value, it would stop counting and the CE signal to the shift registers on Sheet 1 could be forced low. When the first nibble of next packet is received, this CE signal could be set back to a logic high.

This approach assumes that one of the signals from the MII is toggled according to whether the Ethernet chip set is forwarding nibbles through the MII. This signal would be used to reset this counter and allow to begin to count.

The first, unfinished block of control logic was designed to prevent data within an Ethernet packet that was being processed from being falsely recognized as the head of a new packet. This alternate counter implementation would prevent the data from both accepted and ignored Ethernet packets from being falsely detected as a packet header.

8.3.2 Delimiter and Address Verification Control Logic

The control logic that detects the Ethernet delimiter, verifies the device address or broadcast bit is implemented and works correctly. The timing diagram labeled fpga_test1 in Appendix C shows the nibbles being shifted through the shift registers. The Ethernet delimiter and device address are verified and the ADDR_MATCH, E-LOAD, and LOAD

signals transition to a logic high accordingly. The fpga_test2 timing diagram show the wrong address being detected and the ADDR_MATCH signal remaining low. The fpga_test3 timing diagram shows the VALID_PKT signal transitioning high due to the detection of a valid Ethernet delimiter and the broadcast bit being set. Accompanying each timing diagram is the command file used to generate the corresponding results.

8.3.3 Length Counter

The counters that form the Length Counter seen on Sheet 2 are implemented and tested. The fpga_test1 timing diagram shows the Length Counter loaded with the intended Ethernet packet length value from the command file and begin counting. The E-LOAD signal causes the counters to load and the PKT_CNT signal remains high while the Length Counter decrements. The UP counter input is grounded to force the counters to decrement their values.

The clock to these counters is generated by a flip-flop that counts by bytes rather than nibbles. This is necessary because the Length field contains the number of bytes in the data field of the packet.

In the final design, a signal will have to be found to initially clear the counters at power up. This signal must remain high for one clock cycle to reset the counters. Another signal will probably need to be ANDed with the SR_CLK signal input as the clock to the counters. Otherwise, the CLR signal will need to be held high until an Ethernet Packet is first received and the SR_CLK becomes active.

8.3.4 Data Counter

The counters that form the Data Counter seen of Sheet 3 are implemented and tested. The fpga_test1 timing diagram shows the Data Counter loaded with the intended value from the command file and begin counting. The LOAD signal causes the counters to load and the IVDS_PKT signal remains high while the Data Counter decrements. The UP counter input is grounded to force the counters to decrement their values.

The clock to these counters is generated by the same flip-flop above, that counts by bytes rather than nibbles. This is necessary because the IVDS Length field contains the number of bytes in the data field of the IVDS packet.

Similar to the Length Counter, a CLR signal must be implemented in the final design to clear the counters. In simulations, the CLR signal was toggled high for one clock cycle of the SR_CLK to clear the counters.

8.3.5 Inter-Packet Counter

The Inter-Packet Counter is implemented on Sheet 4 of the schematic. When the Data Counter reaches zero and the Enable signal remains a logic high, the Inter-Packet Counter begins incrementing as seen in the fpga_test4 timing diagram. When the Inter-Packet Counter reaches a certain value, the new IVDS packet has shifted into place in the shift registers. Control logic on the Inter-Packet Counter outputs generates and I-LOAD signal and the new IVDS packet is loaded as seen in the fpga_test4 timing diagram.

The control logic at the bottom of Sheet 5 divides the SR_CLK signal into lower frequency clock signals that are used as inputs to the Length Counter and Data Counter, as well as the Inter-Packet Counter. The 2BYTE clock is used in conjunction with MODE bits from the Mode Latch to generate the Data Latch load signal in both the pixel and audio modes. The AUDIO_LOAD, VIDEO_LOAD, and DATA_LOAD signals are shown on the fpga_test5, fpga_test6, fpga_test7, and fpga_test8 timing diagrams toggling with the proper frequencies, each of which is dependent upon the mode.

8.3.6 Mode Latch

The Mode Latch is implemented on Sheet 7. The Mode Latch can be seen to load and hold on its outputs, the appropriate value on the fpga_test5, fpga_test6, fpga_test7, and fpga_test8 timing diagrams.

8.3.7 Data Latch

The Data Latch is implemented on Sheet 7. The Data Latch can in the test_fpga5 timing diagram be seen to load and hold on it outputs. The values seen on the DL_OUT signal in the timing diagram are specified in the associated command file. Work remains incomplete on the control logic to generate the WRITE signal which enables the tri-state buffer on the Data Latch's outputs for the write cycles to both the Buffer Memory and Audio Memory. The WRITE signal controls many of the functions of the logic block within the FPGA.

8.3.8 Pixel Address Counter

The counters forming the Pixel Address Counter are implemented on Sheet 4. The Pixel Address Counter can be seen in the fpga_test5 timing diagram to load and hold on its outputs the value specified in the command file. The clock for this counter is controlled by the WRITE signal and a combination of the LOAD and SR_CLK signals. The WRITE signal allows the Pixel Address Counter to increment its value. The LOAD and SR_CLK signals are needed to clock in the LOAD signal when loading the counter. Work remains incomplete on the control logic to generate the WRITE signal which enables the tri-state buffer on Pixel Address Counter's outputs for the write cycles to the Buffer Memory.

8.3.9 Input Counter

The Input Counter is implemented on Sheet 5. In the final design, a signal will need to be implemented to replace the simulation PWRUP signal that can reset the Input Counter after loading the design into the FPGA. Work remains incomplete on the control logic to generate the WRITE signal which enables the tri-state buffer on Input Counter's outputs for the write cycles to the Buffer Memory.

8.3.10 Output Counter

The Output Counter is implemented on Sheet 5. In the final design, a signal will need to be implemented to replace the simulation PWRUP signal that can reset the Output Counter after loading the design into the FPGA. Work remains incomplete on the control

logic that generates the UPDATE signal which enables the tri-state buffer on Output Counter's outputs for the read cycles from the Buffer Memory.

8.3.11 Audio Input Counter and Audio Output Counter

The Audio Input Counter and the Audio Output Counter are implemented on Sheet 9. The fpga_test9 timing diagram shows the Audio Output Counter value AUDIO_OUT_COUNT, counting up to the constant value AUDIO_IN_COUNT, held by the Audio Input Counter. When the two counter values are the same, a match is made in the Audio Address Comparator and the AUDIO_MATCH signal transitions to a logic low. This low signal disables the CE input of the Audio Output Counter.

This is demonstrated twice in the timing diagram. At the beginning of the simulation, both counters begin incrementing from zero so the AUDIO_MATCH signal is a logic low until AUDIO_LOAD, the clock to the Audio Input Counter has a rising edge. In the operating environment, the clock to the Audio Output Counter has a much lower frequency the clock to the Audio Input Counter. For this simulation, the clock to the Audio Output Counter has been changed to the TEST_CLK signal which will operate at a frequency closer, but still lower than the AUDIO_LOAD signal.

When a match occurs, the AUDIO_MATCH signal remains low until the clock signal to the Audio Input Counter allows it to increment. Then the AUDIO_MATCH signal transitions to a logic high and the Audio Output Counter is allowed to increment on the rising edge of its clock signal. This is demonstrated in the fpga_test9 timing diagram,

when the TEST_CLK signal oscillates and the Audio Output Counter does not increment because the AUDIO_MATCH signal is low.

It should be noted that the portion of the schematic that contains the Audio Memory counters and comparator had to be isolated from the rest of the schematic in order to demonstrate its functionality. Attached with the timing diagram and command file is the "Special Audio Test Schematic" used to generate the timing diagram.

In the final implementation a signal must be found to act as the CLR signal that resets the counters at the beginning of operation.

8.3.12 Audio Address Comparator

The Audio Address Comparator is implemented on Sheet 9. As described in the previous section, the Audio Address Comparator is shown functioning as specified in the fgpa_test9 timing diagram.

8.3.13 Audio Time Counter

The Audio Time Counter is implemented on Sheet 9. The fpga_test10 timing diagrams generated by associated command file, show the many clock cycles of the timing diagrams show that the Audio Time Counter uses the 13_5 MHZ signal to count up to hexadecimal 262 (610 decimal). The period of the 13_5 MHZ signal is 74.074 ns. Thus the frequency of the 22_1KHZ signal is 22.0949KHz. This 22_1KHZ signal acts as a clock to the Audio Output Counter as shown in the schematic.

8.3.14 NTSC Address Counter

The NTSC Address Counter is implemented on Sheet 8 of the schematic. The fpga_test11 timing diagram shows the NTSC Address Counter value, NTSC_COUNT, incrementing with the 74.074 ns clock. Data is initially placed on the A/V_DATA signal which serves as the input to the Update Latch. On the first falling edge of the UPDATE signal, the data is loaded into the Update Latch as appears on it output lines, OUT_UR.

When the NTSC_COUNT value is the same as PIXELADDR, the value on the Buffer Memory side of the Pixel Address Comparator, the NTSC_MATCH signal transition to a logic high. When this happens, the value held by the Update Latch is released onto the NTSC_DATA bus as shown in the timing diagram for that clock cycle.

8.3.15 Pixel Address Comparator and Update Latch

The Pixel Address Comparator and Update Latch are implemented on Sheet 8 of the schematic. As described above, the Update Latch is shown loading a value and releasing it onto the NTSC Data bus as controlled by the Pixel Address Comparator.

In the final design, a latch must be added to the Pixel Address Comparator input from the Buffer Memory. The pixel address will need to be held in the Pixel Address Comparator so that it is available in during periods in which the data present on that bus is not intended to be used by the Pixel Address Comparator.

Additionally, depending on the total time required to read an 8-bit pixel from the Buffer Memory, convert it into 24-bits using the PROMs, and release it onto the NTSC

Data bus, the one clock cycle delay added by the flip-flops on Sheet 9 may need to be removed.

8.4 Incomplete Portions of the Logic Design

Workview provides the ability to specify pin numbers on the external FPGA to tags within the logic design that represent pins. A table in Appendix D shows the pin number assignments that are to be made for the FPGA on the prototype board design. These assignments have not been made within the logic design. The control signals and associated logic need to be laid out that control the board-level devices that the FPGA interacts with.

8.5 Conclusion

This chapter has presented the current status of the logic design. Timing diagrams were used to show the current functionality of the design based on a set of input assumptions. The status of each of the functional logic blocks was presented with references to the timing diagrams and incomplete portions where appropriate. Most of the logic design has been completed and remaining work is in control logic development, control signal generation to external board-level devices, and explicit pin number assignments in the logic design.

Chapter 9. Conclusion

Interactive Video Data Services require a bi-directional path from the cable system head end and transaction processor to and from the user's home. This IVDS project uses the standard television broadcast as the path to the user. Audio codes are embedded in the NTSC sub-carrier. These audio codes are detected and processed by the user's Audio Link receiver. The return path for packetized user input to the head end is a wireless channel to the Repeater Unit that has been developed as part of this project. The packets are then forwarded on to a transaction processor.

The IVDS Browser System group's research has involved adding a Web Browser System to the core IVDS System to provide World Wide Web browsing capabilities to users, requiring only the use of their Audio Link. The bandwidth of a particular television channel will have to be reserved by the cable service provider for the browsing function. The Repeater will forward user "browsing session" packets to the IVDS Browser System. The Browser System will retrieve the desired web-based content, convert it into the NTSC format, and modulate it onto the bandwidth of the "web channel" reserved by the cable provider.

This thesis has introduced the IVDS System Project and the requirements of the IVDS Browser System. Four Design Concepts were examined, discussing the system designs, available development solutions, the issues involved in development, and the reason why the research was directed away from each particular design concept.

The final Browser System design was presented at the system level and then the Decoder Unit was specified at the board-level. The design specification at a functional

block level was introduced for the FPGA which served as the controller for the Decoder Unit. Some timing issues and the resulting modifications to the design were then discussed.

Finally, because the work was suspended due to funding problems, the project could not be finished. The current status of the FPGA design was, however, documented and timing simulations were used to demonstrate the functional portions of the design. The functional blocks have been constructed and most of the unfinished work involves the control logic that allows the functional blocks to interact and store the pixel values and audio samples in the memories and then forward them to the NTSC Encoder and the Audio DAC, respectively.

Much of the work involving the FPGA design has been completed. To finish the FPGA design, a prototype Decoder board will need to be made and the design programmed into the FPGA. Testing will need to occur to verify the input assumptions and completely understand the interfaces to the other board level chips. The control logic will then need to be developed and tested.

Browser software will need to be written to exercise the Decoder board across an Ethernet network. Because development was suspended, this software was never completed. However some of the scheduling software has been competed and tested by Theodoros David [19].

In the event that the research is once again funded, work can resume and the IVDS Browser System can be completed. Users will then be able to browse the World Wide

Web using only the Audio Link. The cable stream will serve as the forward path to the user and the wireless channel to the Repeater will serve as the return path from the user.

The IVDS System and IVDS Browser System will provide users with an array of services, from interacting with quiz shows, to ordering products or requesting information, to browsing the World Wide Web with only a single hand held remote control. Furthermore, these IVDS Systems as a whole will provide advertisers and content providers a wealth of new ways to reach consumers and new markets in a cost effective manner. This IVDS System has the potential to revolutionize the way in which information from around the corner, to around the globe is made available to the everyday television viewer.

References

1. Robert D. Hof, “These May Really Be PCs For The Rest Of Us,” Business Week, June 24,1996, http://www.businessweek.com/1996/26/b34813.htm.

2. Tom R. Halfhill, “Inside the Web PC,” Byte, March 1996, http://www.byte.com/art/9603/sec7/art1.htm.

3. Robert B. Russell, “IEEE 802.14 Multimedia Modem Protocol for Hybrid Fiber-Coax Metropolitan Area Networks,” IEEE 802.14 Working Group, June 1997, http://802.14.org/about802/execsum.htm.

4. “Cable Modem Frequently Asked Questions,“ Cox Communications, June 1997, http://www.cox.com/highSpeed/modemfaq.html.

5. Phillip E. Mattison, “Practical Digital Video with Programming Examples in C”, 1994, pp. 373-394, John Wiley and Sons, Inc.

6. Chad Fogg, “MPEG-2 FAQ,” April 1996, http://www.mpeg2.de/doc/mpeg2faq/faq_v38.htm.

7. “Datasheet 1 - Acorn 7500 Development System,” Acorn Computers Limited, 1997, http://www.acorn.com/acorn/technology/sheets/001-7500DS/.

8. “Datasheet 10 - Acorn Networking,” Acorn Computers Limited, 1997, http://www.acorn.com/acorn/technology/sheets/010-Networking/.

9. “Datasheet 17 - TV based applications with ARM 7500,” Acorn Computers Limited, 1997, http://www.acorn.com/acorn/technology/sheets/017-TV/.

10. “Network Computing - an Acorn Perspective,” Acorn Computers Limited, 1997, http://www.acorn.co.uk/acorn/products/nc/qanda.html.

11. “LSI Logic's Integra™ Architecture Makes $300 TV Set-Top Box A Reality Through Coreware® Program,” LSI Logic, March 1996, http://www.lsilogic.com/mediakit/pr0001.html.

12. “Integra™ SDP1000 Development Platform,” LSI Logic, March 1996, http://www.lsilogic.com/products/unit5_7j.html.

13. “INTEGRA™ Architecture Fact Sheet,” LSI Logic, March 1996, http://www.lsilogic.com/products/unit5_7k.html.

14. “L64008 Transport Embedded CPU and Control Integra™ 1000 Product Family,” LSI Logic, March 1996, http://www.lsilogic.com/products/unit5_6g.html.

15. “L64008 Block Diagram,” LSI Logic, March 1996, http://www.lsilogic.com/graphics/l64008.gif.

16. “L64005 Enhanced MPEG2 Audio/Video Decoder Integra™ 1000 Product Family,” LSI Logic, March 1996, http://www.lsilogic.com/products/unit5_6h.html.

17. “L64768 Single-Chip Cable Receiver Integra™ 1000 Product Family,” LSI Logic, March 1996, http://www.lsilogic.com/products/unit5_7h.html.

18. “L64768 Internal Architecture,” LSI Logic, March 1996, http://www.lsilogic.com/graphics/l64768.gif.

19. Theodoros David, “Networking Requirements and Solutions for a TV WWW Browser,” M.S. Thesis, Virginia Polytechnic Institute and State University, 1997.

20. “XC5200 Field Programmable Gate Arrays”, Xilinx, August 1996, pp. 1-6, http://www.xilinx.com/partinfo/5200.pdf.

21. Gerd E. Keiser, *Local Area Networks*, pp. 404-405, McGraw-Hill, 1989.

22. “DP83840 10/100 Mb/s Ethernet Physical Layer”, National Semiconductor, November 1995, pp. 1-6, http://www.national.com/ds/DP/DP83840A.pdf.

Appendix A: Buffer Memory Delay Calculations

Assume that the Browser wants to refresh the entire screen that the user is viewing. The Browser sends a series of Ethernet packets each containing a 1500-byte data field. Within the data field of each Ethernet packet is a single IVDS packet consisting of color pixel data. Each IVDS data packet contains 6 bytes of overhead. The remaining 1494 bytes are either 498 24-bit color pixels or 1494 8-bit color pixels.

$$(\frac{1494 bytes}{24 bits/pixel})(\frac{8 bits}{byte}) = 498 pixels$$
$$(\frac{1494 bytes}{8 bits/pixel})(\frac{8 bits}{byte}) = 1494 pixels$$

In 24-bit color mode, the Decoder receives a new pixel every 240ns.

$$\frac{24 bits/pixel}{100e6 bits/\sec} = 240 ns/pixel$$

In 8-bit color mode, the Decoder receives a new pixel every 80ns.

$$\frac{8 bits/pixel}{100e6 bits/\sec} = 80 ns/pixel$$

The NTSC encoder uses a 13.5 MHz clock and thus expects to receive a 24-bit RGB pixel color every 74.074ns.

$$\frac{1}{13.5 MHz} = 74.074 ns$$

The Ethernet interframe time is 9.6µs.(reference) There are 19 bytes of header and trailer data and a maximum of 1500 bytes of data in one Ethernet packet. The time to receive one Ethernet packet is then 131.12µs.

$$\frac{8\,bits/byte(19+1500)bytes}{100x10^6\ bits/\sec ond} + 9.6x10^{-6} = 131.12x10^{-6}\ \sec onds$$

In order to fill the 28k Buffer Memory space with 24-bit color pixels, it will require 56.22488996 Ethernet packets. In 8-bit pixel mode, 18.7416332 Ethernet packets will be required to fill the Buffer Memory space.

$$\frac{28{,}000\,pixels}{498\ pixels/packet} = 56.2248996\,packets$$

$$\frac{28{,}000\,pixels}{1494\ pixels/packet} = 18.7416332\,packets$$

Thus, the time required to fill the 28k Buffer Memory is the time to receive the number of complete packets required to fill up the memory plus the time to receive the Ethernet header and fraction of the final Ethernet packet.

Time to fill the 28K Buffer Memory using 8-bit pixels:

$$(131.12x10^{-6}\ \sec)(18) + \frac{(8\,bits/byte)(21bytes + 0.7416332(1494bytes)}{100x10^6\ bits/\sec} = 2.4488ms$$

Time to fill the 28K Buffer Memory using 24-bit pixels:

$$(131.12x10^{-6}\ \sec)(56) + \frac{(8\,bits/byte)(21bytes + 0.2248996(1494bytes)}{100x10^6\ bits/\sec} = 7.37128ms$$

The 74.074ns NTSC clock will be used to read pixel values from the Buffer Memory. The time required to read every pixel from the 28K Buffer Memory is 2.074ms.

$$(28{,}000\,pixels)(74.074\,ns/pixel) = 2.074ms$$

Thus, pixels from the Buffer Memory can be read and stored in the Main Memory at a faster rate than they can be received and stored in the Buffer Memory. This result applies to both the 8-bit and 24-bit modes.

Using a Buffer Memory space size of 28K, the Buffer Memory will need to be loaded and emptied 10.97 times to refresh an entire screen of 307,200 pixels.

$$\frac{307{,}200\,pixels}{28{,}000\,pixels} = 10.97$$

When the very first pixel color is ready to be written into the Buffer Memory, the NTSC Address Counter has an equal probability of being at any of the 307,200 possible 640x480 screen pixel locations. The new pixel color and its corresponding screen location address, X, is stored in the Buffer Memory. It will take between roughly 2.45ms and 7.37ms to fill the Buffer Memory.

The NTSC Encoder refreshes the television screen 30 times per second (every 33.3ms) using the pixels stored in the Main Memory. The Browser has no way of knowing where the current pixel being refreshed to the television screen is. In the worst case the first pixel must wait 33.3ms before it is copied into the Main Memory.

After waiting 33.3ms the Browser can send another 28,000 pixels to the Decoder. The Browser does not have to worry about overwriting pixels in the Buffer Memory for two reasons. First, waiting the 33.3ms assures that the NTSC Address Counter has cycled completely through the Main Memory. Second, the NTSC subsystem can copy pixels into the Main Memory faster than they can be received and stored in the Buffer Memory. The Decoder will receive a new 8-bit color pixel every 80ns and a new 24-bit color pixel every 240ns. The NTSC clock will allow a pixel to be read every 74.074ns.

The Browser will wait 33.3ms before sending each group of 28,000 pixels. This will assure that the NTSC Address Counter has cycled through the Main Memory and the previous group has begun to copied in the Main Memory. The time required to copy

every pixel from the Buffer Memory in the Main Memory is 2.074ms which is far less than the 33.3ms required to cycle through that memory.

Thus, the time to refresh the entire screen is dependent upon the number of times the Buffer Memory must be filled to have 307,200 new pixels copied in the Main Memory. The Buffer Memory can store 28,000 pixels so it takes eleven groups of these 28,000 pixels to refresh the screen. The Browser waits 33.3ms per group of 28,000 pixels so the time required to refresh the screen is 0.367 seconds, roughly a third of a second.

$$\frac{1}{30}(11) = 0.367\,\text{sec}onds$$

The IVDS Browser System could refresh the entire screen that the user was viewing in roughly a third of second using a group of 32Kx8 SRAMs as the Buffer Memory in the Decoder. This is a greatly simplified assumption however, because the other system delays are not taken into account.

Furthermore, it would be very difficult to estimate what the sum of these delays would be. The sponsor therefore wanted provide the ability to, at a later time, reduce the third of a second refresh time involving the Buffer Memory. The Decoder board design was modified to provide sockets to hold another group of six 32Kx8 SRAMs, effectively doubling the size of the Buffer Memory. The FPGA design was modified to be able to address the larger 64Kx8 memory space. A jumper would be added to the board to identify to the FPGA when the second group of SRAMs were added to the board.

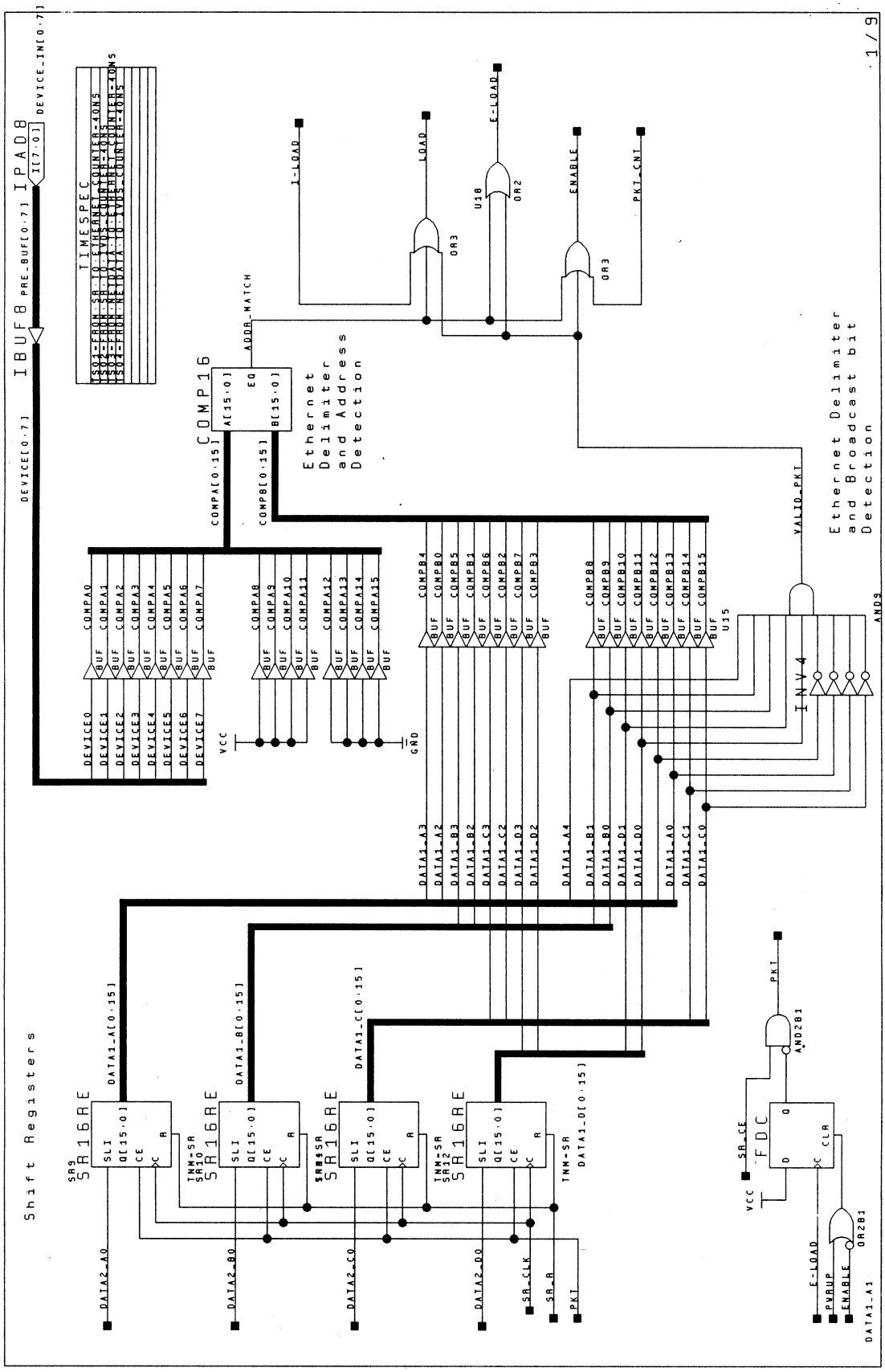
Adding this additional group of SRAMs effectively doubled the size of the Buffer Memory. The lowest 4,000 48-bit memory slices would be dedicated to the palette storage. And the upper 60,000 slices would buffer the incoming pixels. The Browser

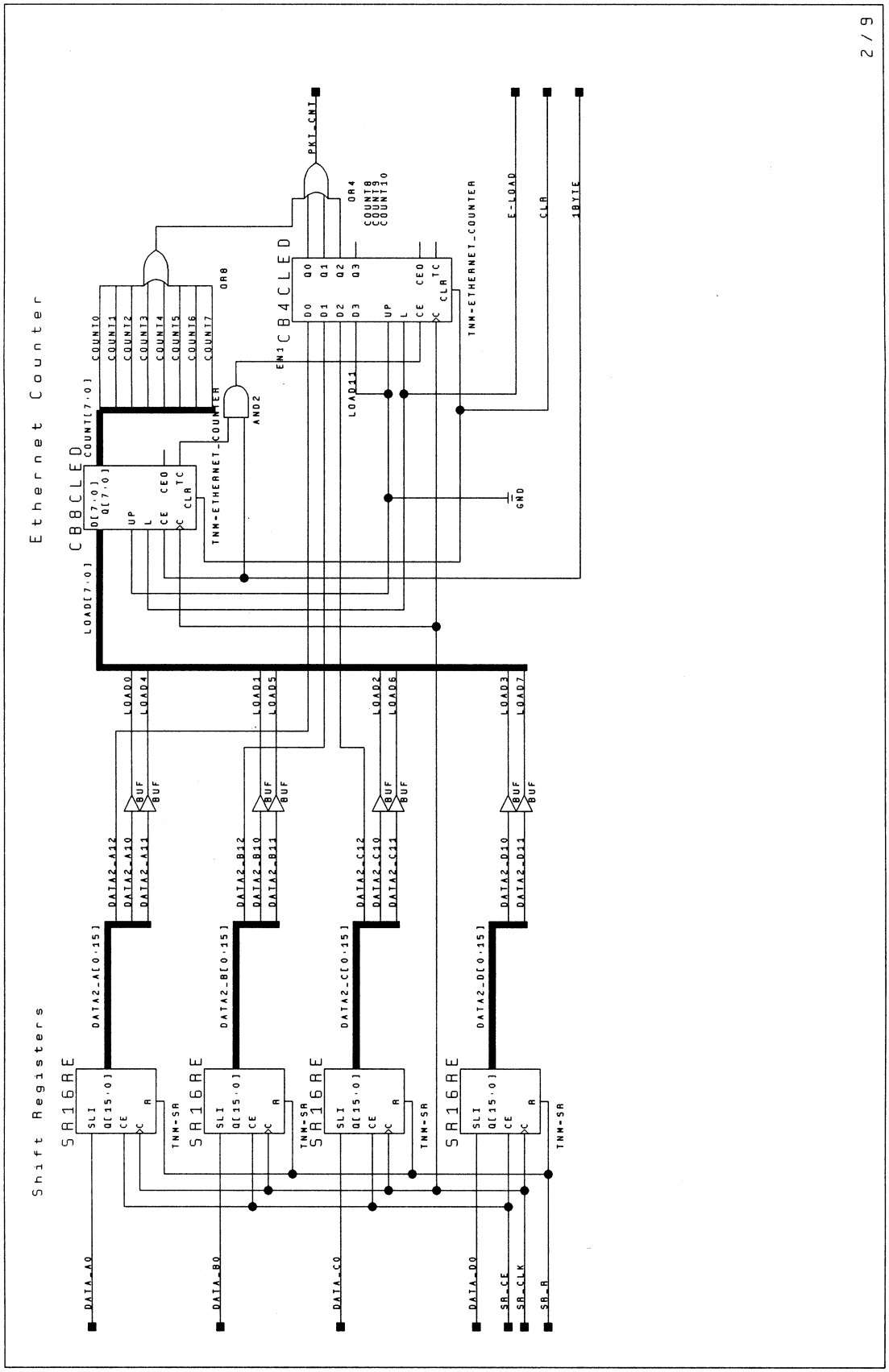
would be able to send pixels in groups of 60,000 while still waiting 33.3ms between group transmissions. The times involved in storing and copies pixels to and from the Buffer Memory would roughly double and still remain far below the 33.3ms NTSC Address Counter cycle time.

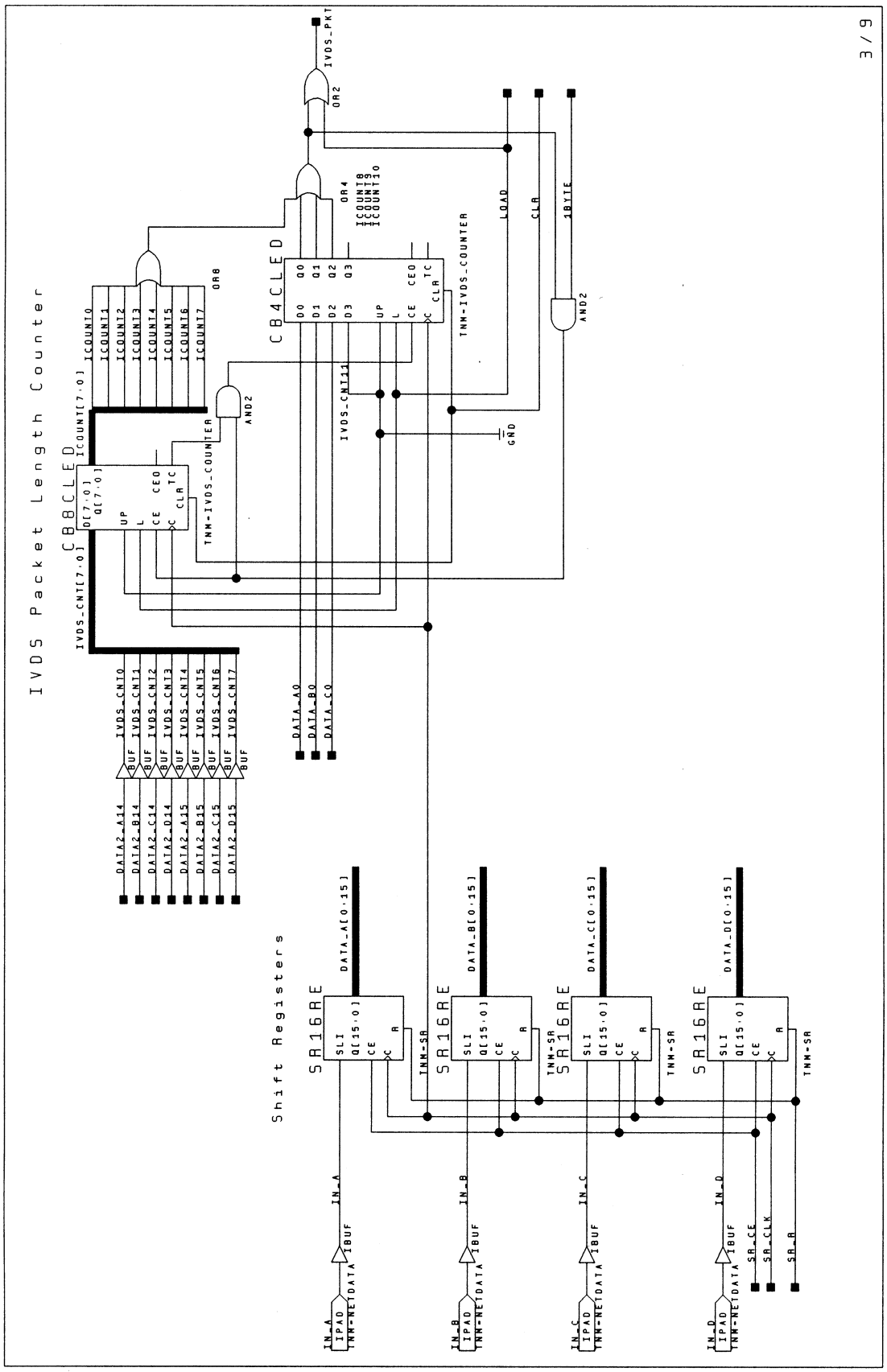
$$\frac{307{,}200\,pixels}{60{,}000\,pixels} = 5.12 \cong 6$$

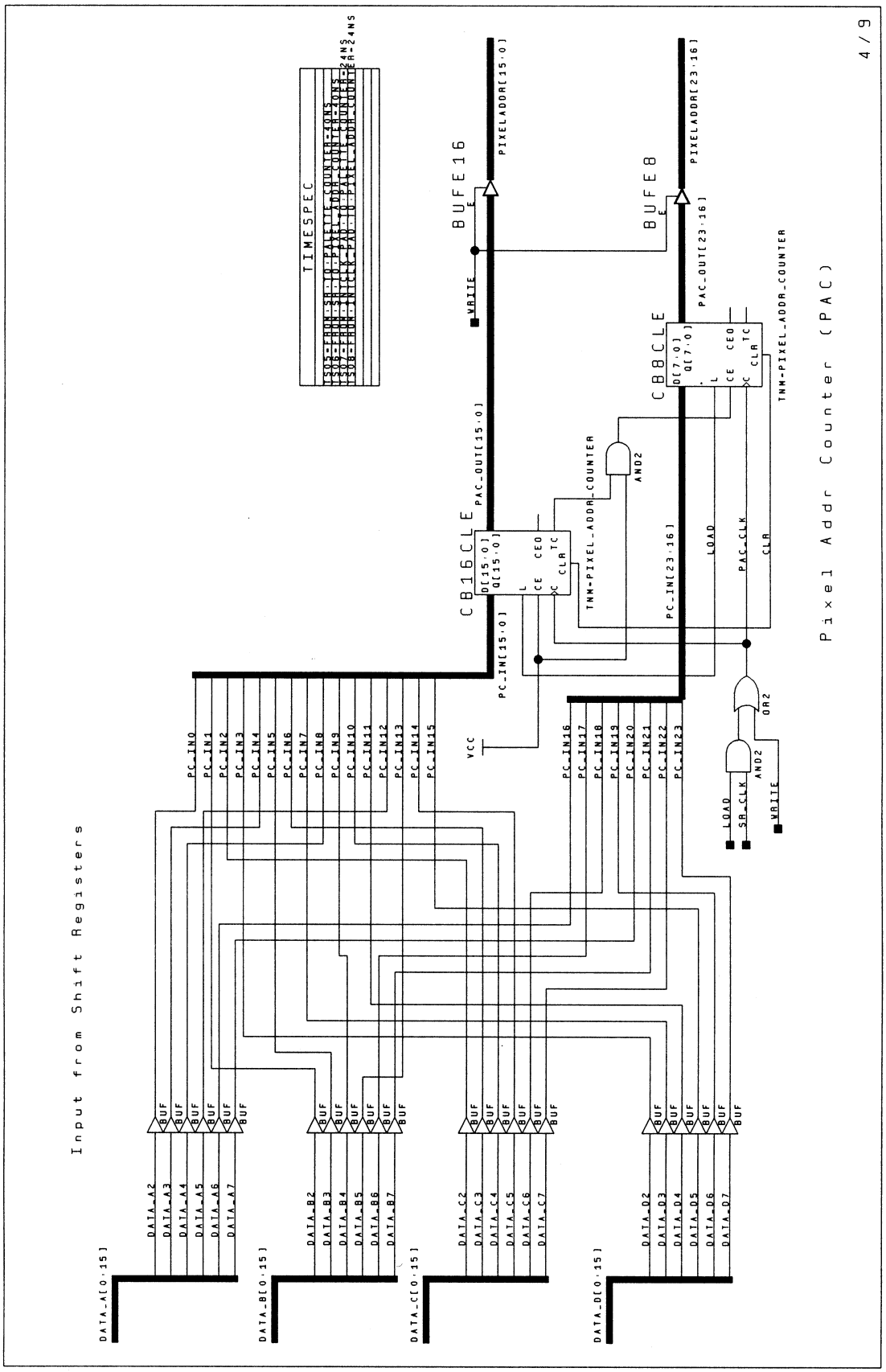
$$\tfrac{1}{30}(6) = 0.200\,\text{sec}onds$$

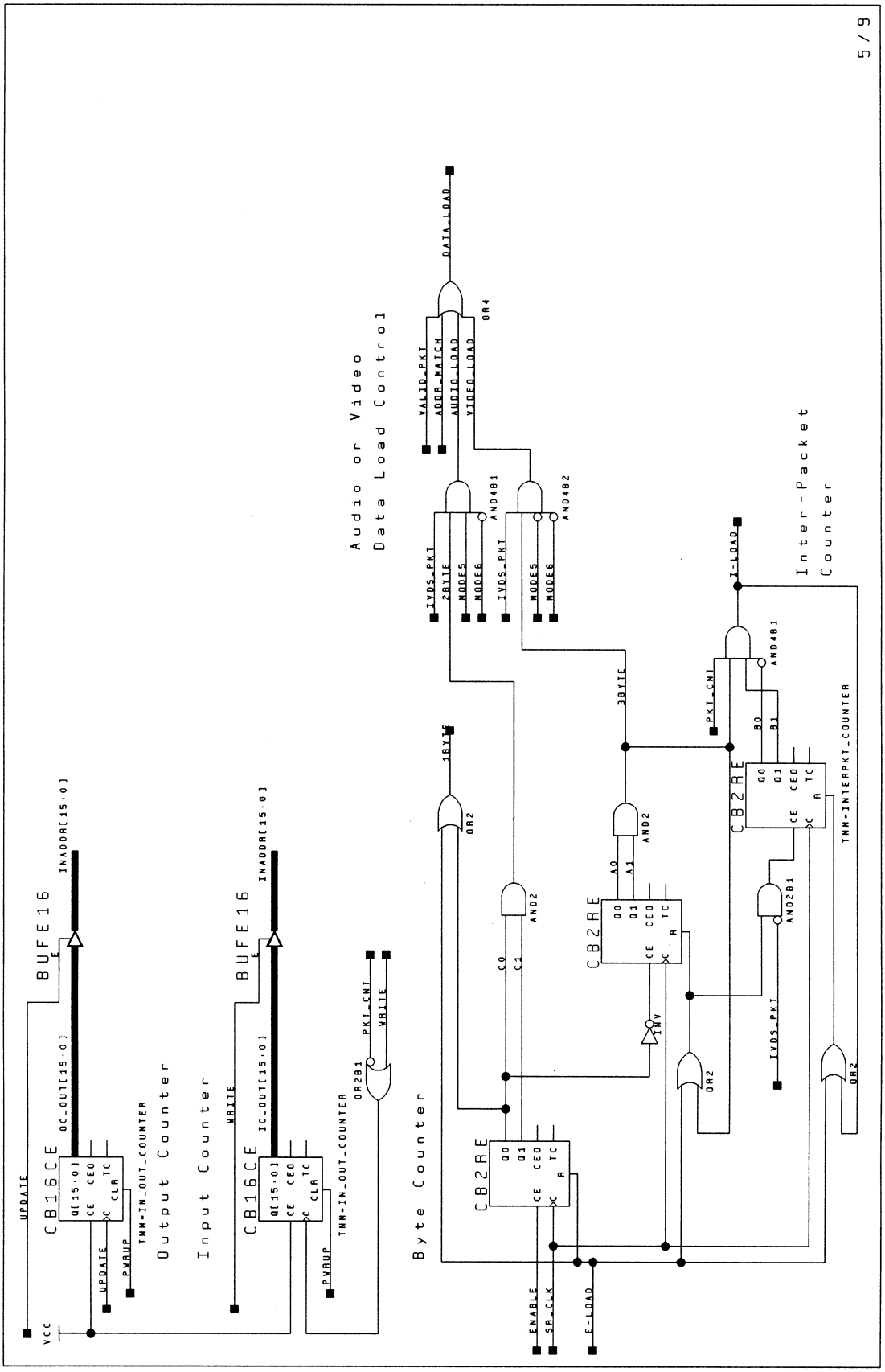
Appendix B: FPGA Design Schematic


Shift Registers
Ethernet Delimiter and Address Detection
Ethernet Delimiter and Broadcast bit Detection
IBUF8
IPAD8
COMP16
SR16RE
FDC
INV4
TIMESPEC
1/9

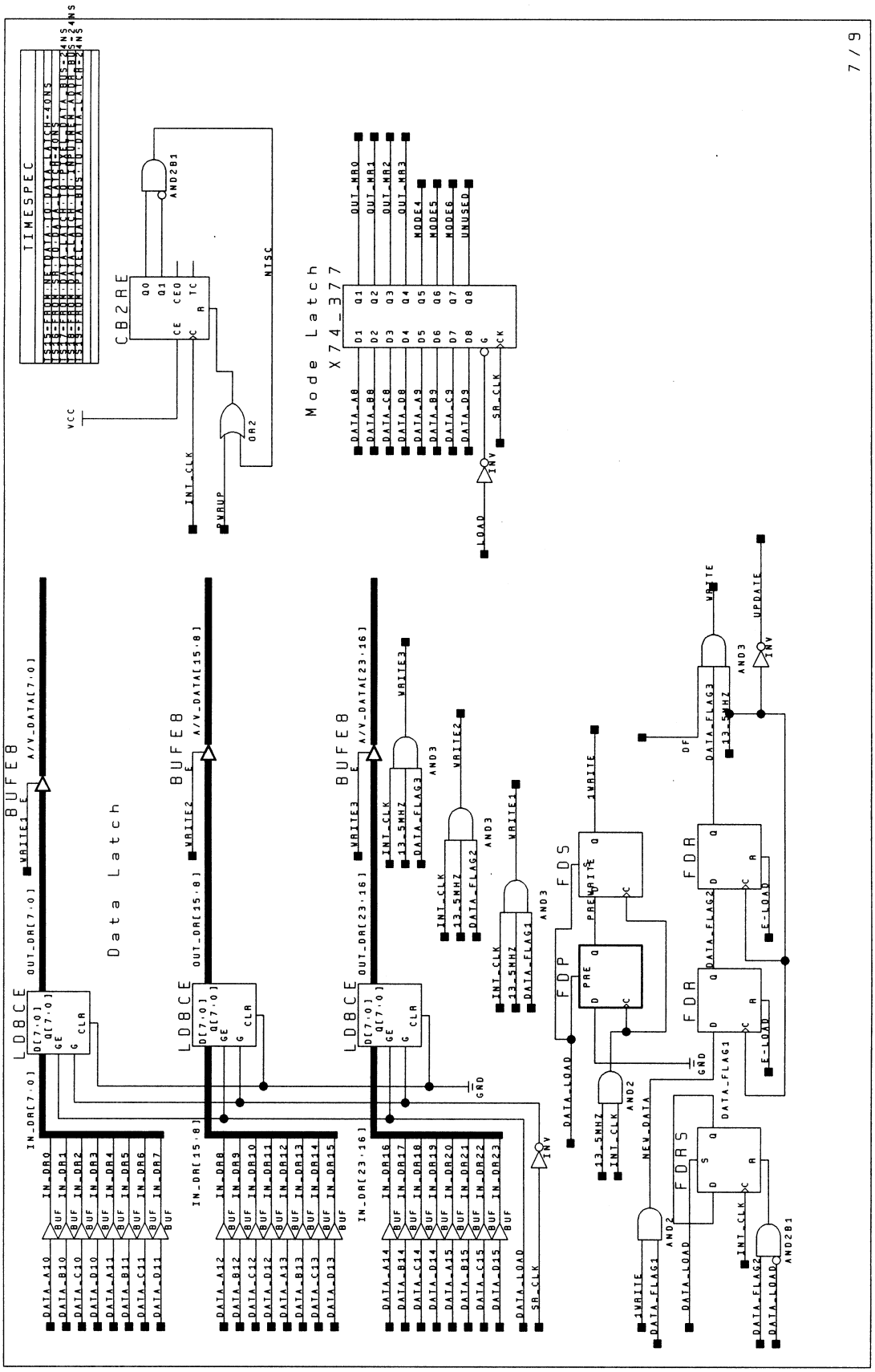

2 / 9
Ethernet Counter
Shift Registers
CB8CLED
D[7:0]
Q[7:0]
UP
L
CE
CLR
CEO
TC
COUNT[7:0]
COUNT0
COUNT1
COUNT2
COUNT3
COUNT4
COUNT5
COUNT6
COUNT7
OR8
AND2
TNM-ETHERNET_COUNTER
LOAD[7:0]
EN1CB4CLED
Q0
Q1
Q2
Q3
D0
D1
D2
D3
OR4
COUNT8
COUNT9
COUNT10
PKT_CNT
LOAD11
GND
E-LOAD
CLR
18TE
LOAD0
LOAD4
LOAD1
LOAD5
LOAD2
LOAD6
LOAD3
LOAD7
BUF
DATA2_A12
DATA2_A10
DATA2_A11
DATA2_B12
DATA2_B10
DATA2_B11
DATA2_C12
DATA2_C10
DATA2_C11
DATA2_D10
DATA2_D11
DATA2_A[0:15]
DATA2_B[0:15]
DATA2_C[0:15]
DATA2_D[0:15]
SR16RE
SLI
Q[15:0]
CE
R
TNM-SR
DATA_A0
DATA_B0
DATA_C0
DATA_D0
SR_CE
SR_CLK
SR_R

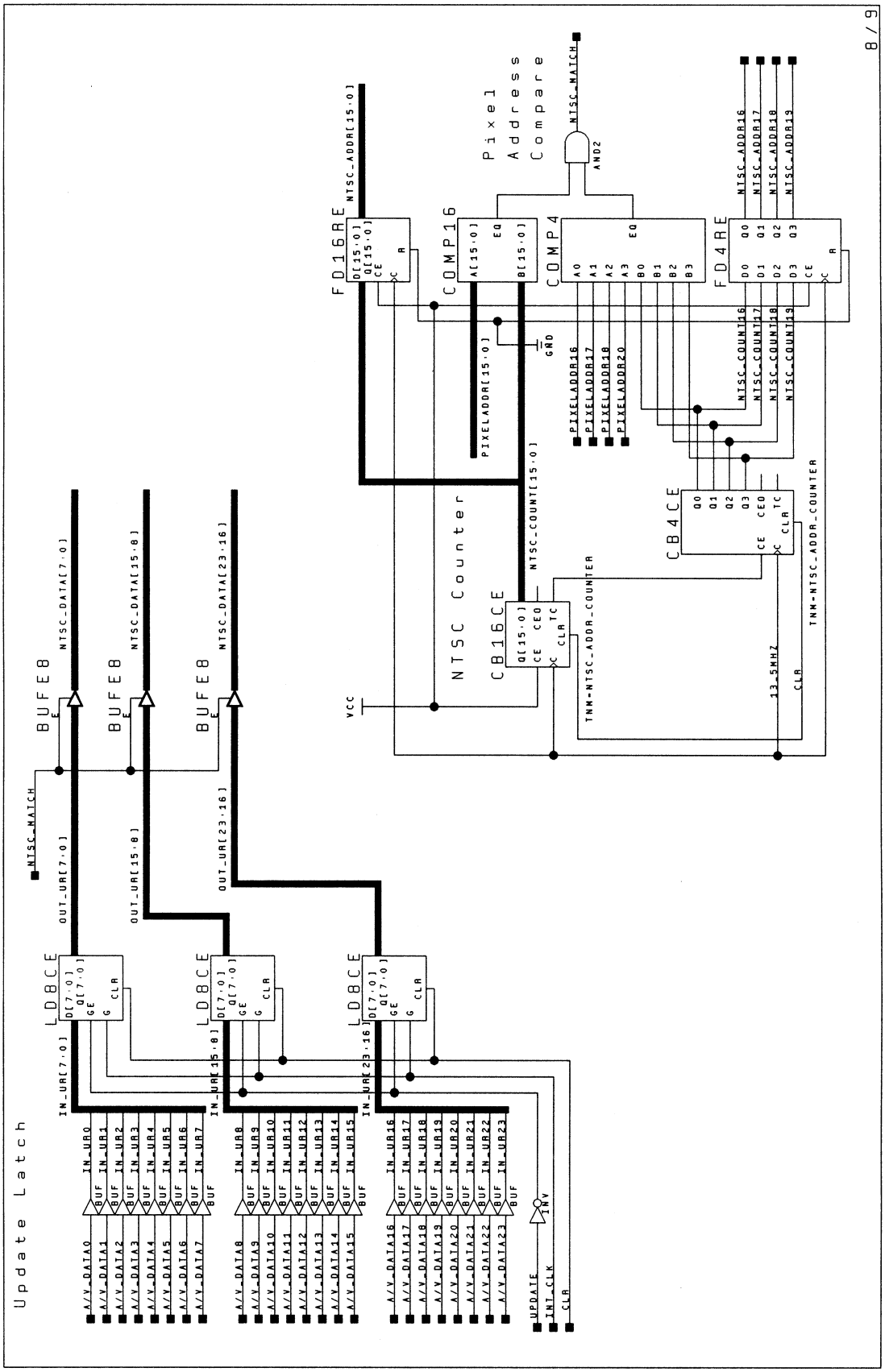

IVDS Packet Length Counter
3/9
CB8CLED
CB4CLED
SR16RE
Shift Registers
IVDS_PKT
OR2
OR4
OR8
AND2
LOAD
CLR
1BYTE
GND
TNM-IVDS_COUNTER
IVDS_CNT11
ICOUNT[7:0]
IVDS_CNT[7:0]
DATA_A0
DATA_B0
DATA_C0
DATA_A[0:15]
DATA_B[0:15]
DATA_C[0:15]
DATA_D[0:15]
IN_A
IN_B
IN_C
IN_D
IPAD
IBUF
TNM-SR
SR_CE
SR_CLK
SR_R

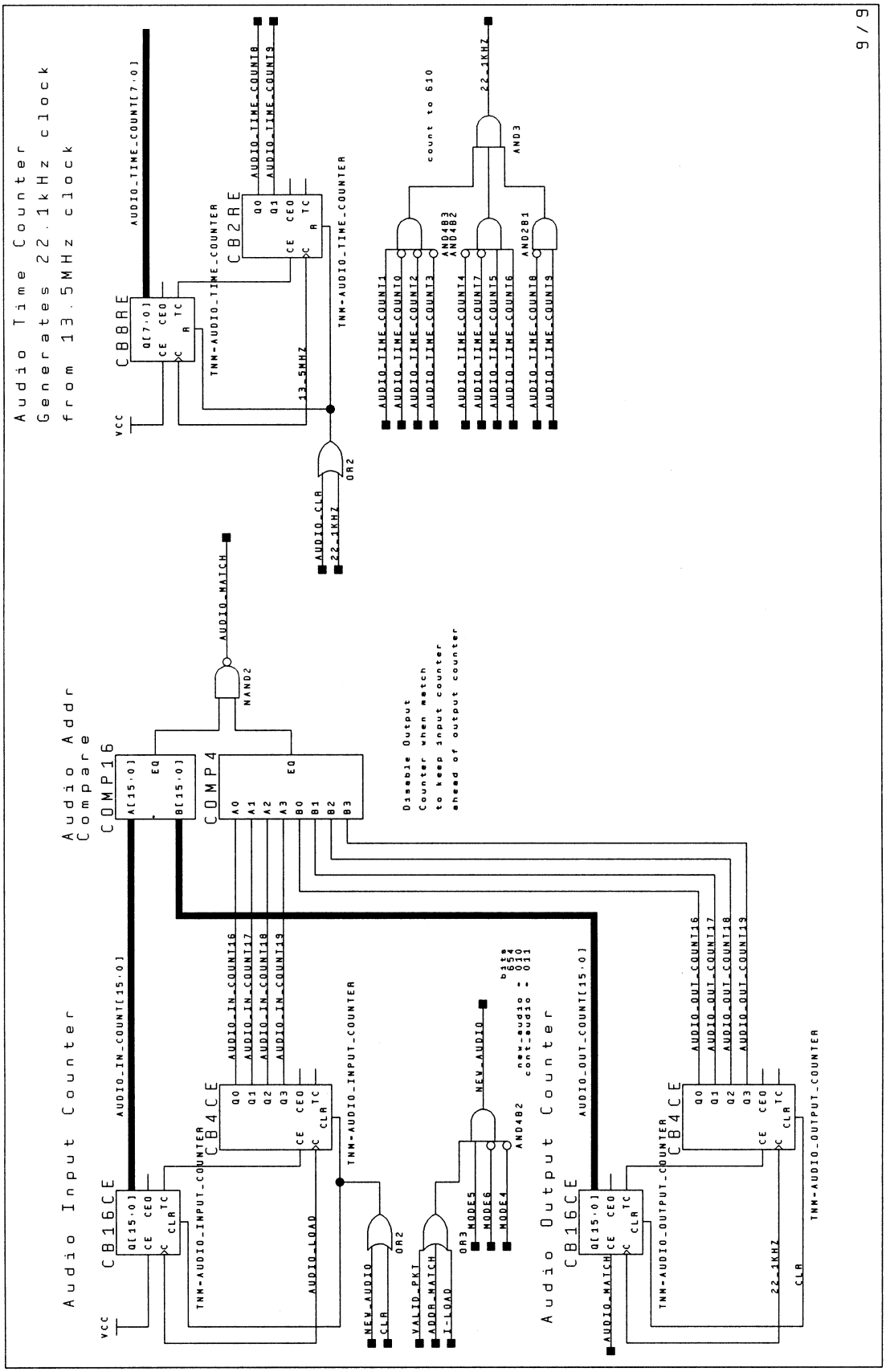

Pixel Addr Counter (PAC)
4/9
Input from Shift Registers
TIMESPEC
BUFE16
BUFE8
CB16CLE
CB8CLE
PIXELADDR[15:0]
PIXELADDR[23:16]
PAC_OUT[15:0]
PAC_OUT[23:16]
PC_IN[15:0]
PC_IN[23:16]
TNM-PIXEL_ADDR_COUNTER
AND2
OR2
LOAD
PAC_CLK
CLR
WRITE
SR_CLK
VCC
DATA_A[0:15]
DATA_B[0:15]
DATA_C[0:15]
DATA_D[0:15]
BUF


5/9
Output Counter
Input Counter
Byte Counter
Audio or Video
Data Load Control
Inter-Packet
Counter
CB16CE
BUFE16
CB2RE
AND2
OR2
AND4B1
AND2B1
INV
OR4
OR2B1
VCC
UPDATE
PXRUP
TNM-IN_OUT_COUNTER
OC_OUT[15:0]
IC_OUT[15:0]
INADDR[15:0]
WRITE
PKT_CNT
TNM-INTERPKT_COUNTER
ENABLE
SR_CLK
E-LOAD
I-LOAD
DATA_LOAD
VALID_PKT
ADDR_MATCH
AUDIO_LOAD
VIDEO_LOAD
IVDS_PKT
2BYTE
MODE5
MODE6
3BYTE
1BYTE


OBUF16
DATABUS1[15:0]
IOPAD16
IO[15:0]
A/V_DATA[15:0]
TNM-PIXEL_DATA_BUS
Data Interface
to pixel data half
of Input Memory
A/V_DATA[23:16]
OBUF8
IOPAD8
IO[7:0]
DATABUS1[23:16]
TNM-PIXEL_DATA_BUS
TIMESPEC
TS05=FROM:IN_OUT_COUNTER:TO:INPUTMEM_ADDR_BUS=24NS
TS10=FROM:PALETTE_COUNTER:TO:INPUTMEM_ADDR_BUS=24NS
TS11=FROM:PIXEL_ADDR_COUNTER:TO:PIXEL_ADDR_BUS=24NS
TS12=FROM:MODE_LATCH:TO:INPUTMEM_ADDR_BUS=24NS
TS13=FROM:INITCLK_PAD:TO:FF5=24NS
PIXELADDR[15:0]
OBUF16
DATABUS2[15:0]
IOPAD16
IO[15:0]
TNM-PIXEL_ADDR_BUS
Data Interface
to pixel addr half
of Input Memory
PIXELADDR[23:16]
OBUF8
IOPAD8
IO[7:0]
DATABUS2[23:16]
TNM-PIXEL_ADDR_BUS
INADDR[15:0]
OBUF16
OPAD16
O[15:0]
ADDRBUS1[15:0]
TNM-INPUTMEM_ADDR_BUS
Address Interface
to Input Memory
13_5MHZ
IPAD
IBUF
13_5MHZ
PKT_HEAD
IPAD
IBUF
PKT_HEAD
INT_CLK
IPAD
IBUF
BUFG
INT_CLK
TNM-INITCLK_PAD
SA_CE
IPAD
IBUF
BUFG
SA_CE
TNM-NET DATA
SA_R
IPAD
IBUF
SA_R
SA_CLK
IPAD
IBUF
BUFG
SA_CLK
TNM-NET DATA
REG_CLR
IPAD
IBUF
BUFG
REG_CLR
TNM-NET DATA
CLR
IPAD
IBUF
CLR
6 / 9


TIMESPEC
CB2RE
Mode Latch
X74_377
Data Latch
BUFE8
LDBCE
FDP
FDS
FDR
FDRS
AND2
AND3
AND2B1
OR2
INV
BUF
VCC
GND
INT_CLK
VRUP
NTSC
LOAD
SR_CLK
OUT_MR0
OUT_MR1
OUT_MR2
OUT_MR3
MODE4
MODE5
MODE6
UNUSED
DATA_A8
DATA_B8
DATA_C8
DATA_D8
DATA_A9
DATA_B9
DATA_C9
DATA_D9
A/V_DATA[7:0]
A/V_DATA[15:8]
A/V_DATA[23:16]
OUT_DR[7:0]
OUT_DR[15:8]
OUT_DR[23:16]
IN_DR[7:0]
IN_DR[15:8]
IN_DR[23:16]
VRITE1
VRITE2
VRITE3
13.5MHZ
DATA_FLAG1
DATA_FLAG2
DATA_FLAG3
DATA_LOAD
NEW_DATA
1VRITE
VRITE
UPDATE
DF
E-LOAD
7 / 9


Update Latch
8/9
Pixel Address Compare
NTSC Counter
FD16RE
COMP16
COMP4
FD4RE
CB16CE
CB4CE
LDBCE
BUFE8
AND2
INV
BUF
NTSC_ADDR[15:0]
NTSC_MATCH
PIXELADDR[15:0]
NTSC_COUNT[15:0]
PIXELADDR16
PIXELADDR17
PIXELADDR18
PIXELADDR20
NTSC_ADDR16
NTSC_ADDR17
NTSC_ADDR18
NTSC_ADDR19
NTSC_COUNT16
NTSC_COUNT17
NTSC_COUNT18
NTSC_COUNT19
TNM-NTSC_ADDR_COUNTER
13.5MHZ
CLR
VCC
GND
NTSC_DATA[7:0]
NTSC_DATA[15:8]
NTSC_DATA[23:16]
OUT_UR[7:0]
OUT_UR[15:8]
OUT_UR[23:16]
IN_UR[7:0]
IN_UR[15:8]
IN_UR[23:16]
A/V_DATA0
A/V_DATA1
A/V_DATA2
A/V_DATA3
A/V_DATA4
A/V_DATA5
A/V_DATA6
A/V_DATA7
A/V_DATA8
A/V_DATA9
A/V_DATA10
A/V_DATA11
A/V_DATA12
A/V_DATA13
A/V_DATA14
A/V_DATA15
A/V_DATA16
A/V_DATA17
A/V_DATA18
A/V_DATA19
A/V_DATA20
A/V_DATA21
A/V_DATA22
A/V_DATA23
UPDATE
INT_CLK
CLR

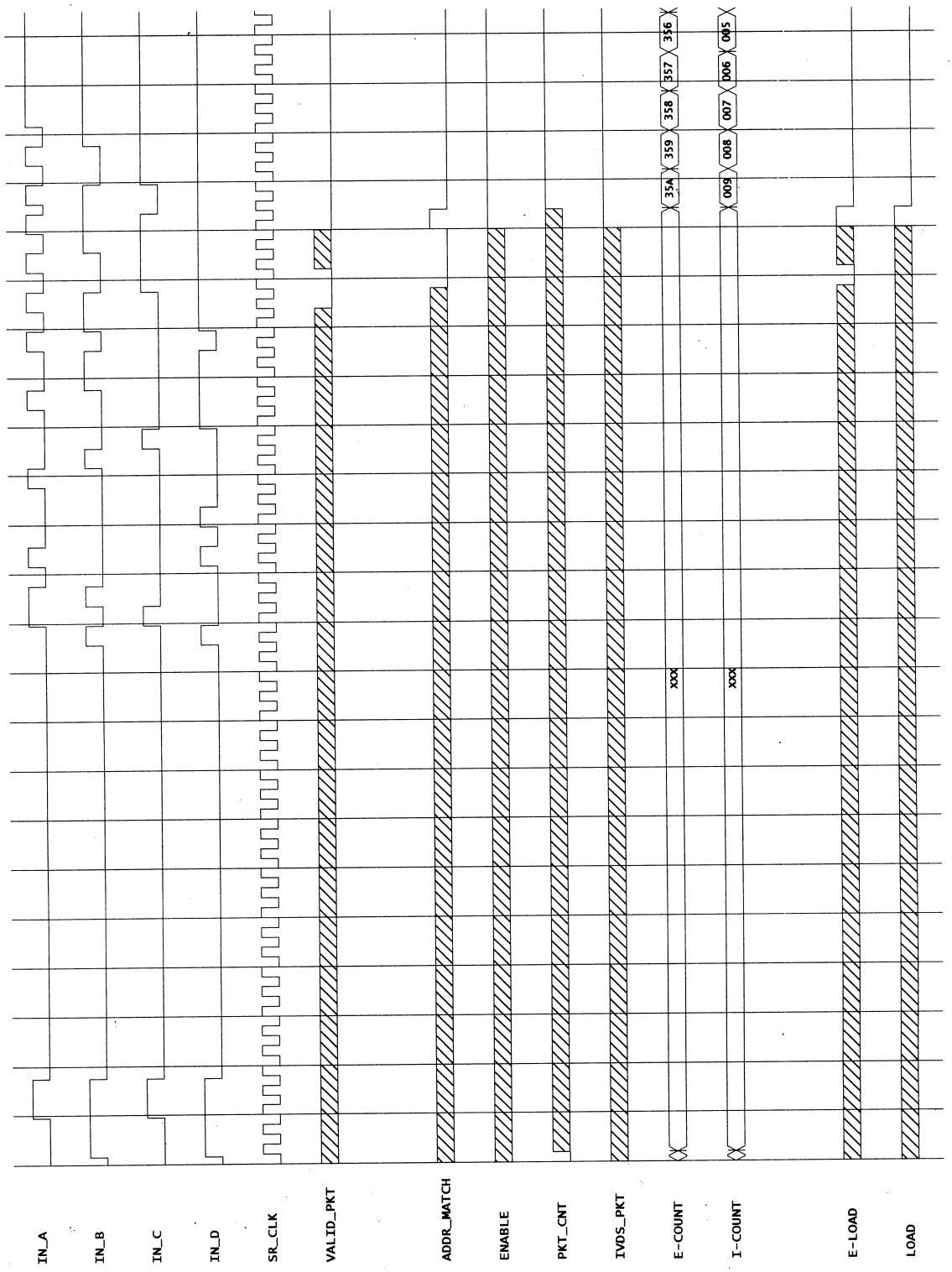

Audio Time Counter
Generates 22.1kHz clock
from 13.5MHz clock
9 / 9
CB8RE
CB2RE
AUDIO_TIME_COUNT[7:0]
TNM-AUDIO_TIME_COUNTER
13.5MHZ
AUDIO_TIME_COUNT8
AUDIO_TIME_COUNT9
TNM-AUDIO_TIME_COUNTER
count to 610
22.1KHZ
AND3
AND4B3
AND4B2
AND2B1
OR2
AUDIO_CLR
22.1KHZ
AUDIO_MATCH
NAND2
Audio Addr
Compare
COMP16
COMP4
Disable Output
Counter when match
to keep input counter
ahead of output counter
Audio Input Counter
CB16CE
CB4CE
AUDIO_IN_COUNT[15:0]
AUDIO_IN_COUNT16
AUDIO_IN_COUNT17
AUDIO_IN_COUNT18
AUDIO_IN_COUNT19
TNM-AUDIO_INPUT_COUNTER
AUDIO_LOAD
NEW_AUDIO
VALID_PKT
ADDR_MATCH
I_LOAD
OR3
MODE5
MODE6
MODE4
AND4B2
new_audio = 010
cont_audio = 011
Audio Output Counter
AUDIO_OUT_COUNT[15:0]
AUDIO_OUT_COUNT16
AUDIO_OUT_COUNT17
AUDIO_OUT_COUNT18
AUDIO_OUT_COUNT19
TNM-AUDIO_OUTPUT_COUNTER
22.1KHZ
CLR
VCC

Appendix C: Testing Scripts and Timing Diagrams

C.1 FPGA_TEST1

```
| this file tests the basic ability to match the device addr of all ones
| and match the delimiter of 10101010.
| this test file shifts data into a shift register
| and loads the Ethernet Length and IVDS Length fields into the respective couinters
| when the delimiter and addr match.
| the counters loads with the desired value and begin counting.


net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector ivds_cnt ivds_cnt[7:0]
vector startaddr pac_in[23:16] pac_in[15:0]
vector inaddr inaddr[15:0]
vector pc_out pc_out[15:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector c c1 c0
vector a a1 a0
vector b b1 b0
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector control_cnt control_cnt3 control_cnt2 control_cnt1 control_cnt0
vector mode mode6 mode5 mode4
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]
vector q q3 q2 q1 q0
vector cc cc3 cc2 cc1 cc0

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0

wave fpga_test1.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count pkt  E-load load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb
```

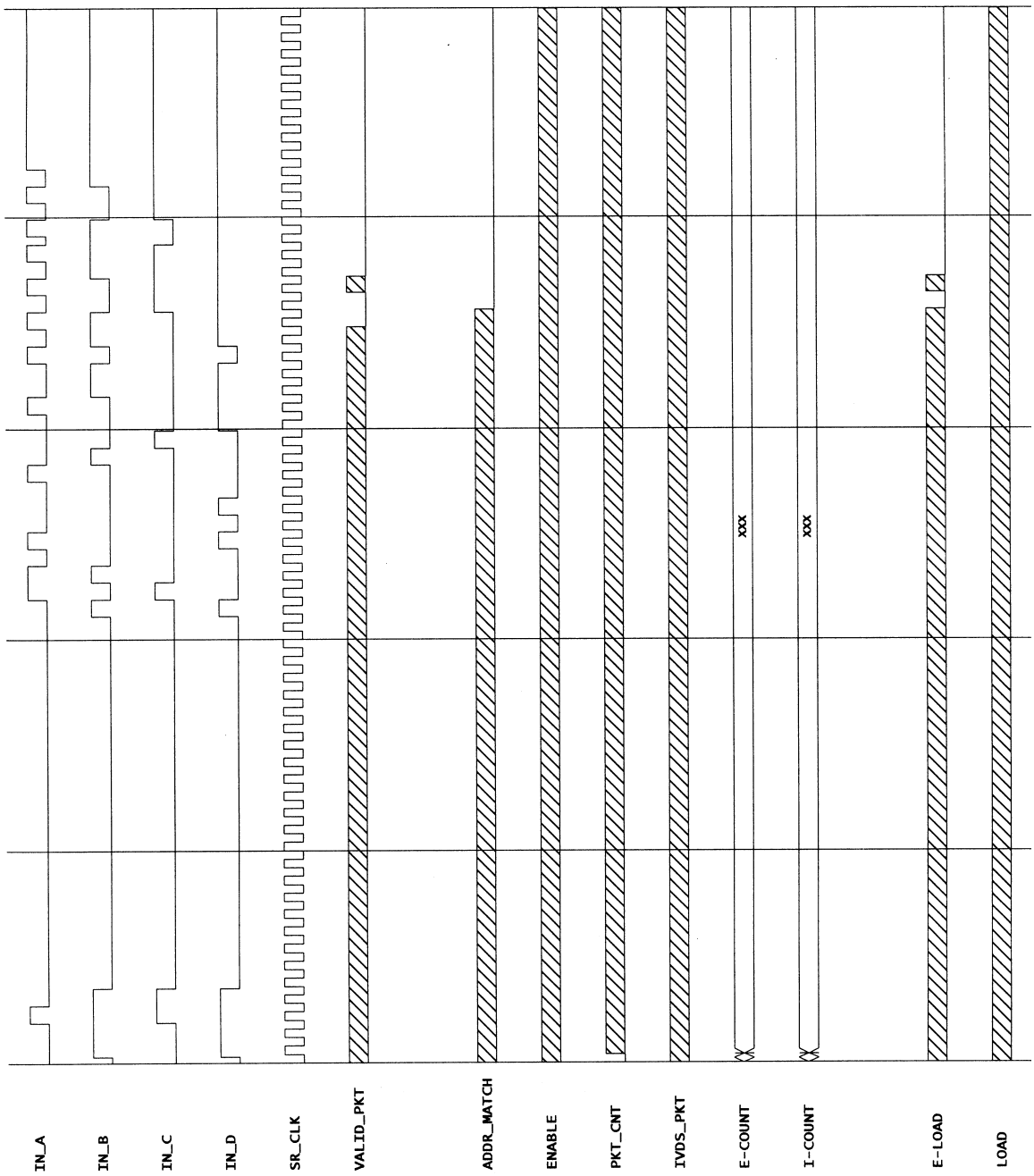
```
| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8        9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 0009
wfm  in_a  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=1 @1335ns=0    | -- -- 7 3
|               14       15        0        1           format

| IVDS Starting Address = A98421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000  palette = A
wfm in_a  @1615ns=0 @1655ns=1         | pallete bit 1        mode bit 1
wfm in_b  @1615ns=1 @1655ns=0         | pallete bit 2        mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3        mode bit 3
wfm in_d  @1615ns=1 @1655ns=0         | pallete bit 4        not used
|                8         9                 format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1 | 20 16 12  8 4 0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1 | 21 17 13  9 5 1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1 | 22 18 14 10 6 2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1 | 23 19 15 11 7 3
|               10       11       12       13       14       15             format

| IVDS DATA = ABCDEF
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1 | 20 16 12  8 4 0
wfm in_b  @1935ns=1 @1955ns=1 @1995ns=0 @2035ns=0 @2075ns=1 @2115ns=1 | 21 17 13  9 5 1
wfm in_c  @1935ns=0 @1955ns=0 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1 | 22 18 14 10 6 2
wfm in_d  @1935ns=1 @1955ns=1 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1 | 23 19 15 11 7 3
|               10       11       12       13       14       15             format

sim 2500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
I-COUNT
E-LOAD
LOAD
XXX
XXX
35A
359
358
357
356
009
008
007
006
005

C.2 FPGA_TEST2

```
| this file tests the basic ability to not match the device addr.
| this test file shifts data into a shift register
| and does not load length counters because the addresses do not match


net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector ivds_cnt ivds_cnt[7:0]
vector startaddr pac_in[23:16] pac_in[15:0]
vector inaddr inaddr[15:0]
vector pc_out pc_out[15:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector c c1 c0
vector a a1 a0
vector b b1 b0
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector control_cnt control_cnt3 control_cnt2 control_cnt1 control_cnt0
vector mode mode6 mode5 mode4
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]
vector q q3 q2 q1 q0
vector cc cc3 cc2 cc1 cc0

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0


wave fpga_test2.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count pkt  E-load load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
```

```
|               0       1

| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8       9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 0009
wfm  in_a  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=1 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = A98421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000  palette = A
wfm in_a  @1615ns=0 @1655ns=1         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=1 @1655ns=0                | pallete bit 2        mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3        mode bit 3
wfm in_d  @1615ns=1 @1655ns=0         | pallete bit 4        not used
|                8         9                 format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1      | 20 16 12  8 4
0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1      | 21 17 13  9 5
1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 22 18 14 10 6
2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = ABCDEF
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 20 16 12  8 4
0
wfm in_b  @1935ns=1 @1955ns=1 @1995ns=0 @2035ns=0 @2075ns=1 @2115ns=1      | 21 17 13  9 5
1
wfm in_c  @1935ns=0 @1955ns=0 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 22 18 14 10 6
2
wfm in_d  @1935ns=1 @1955ns=1 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

sim 2500ns
```

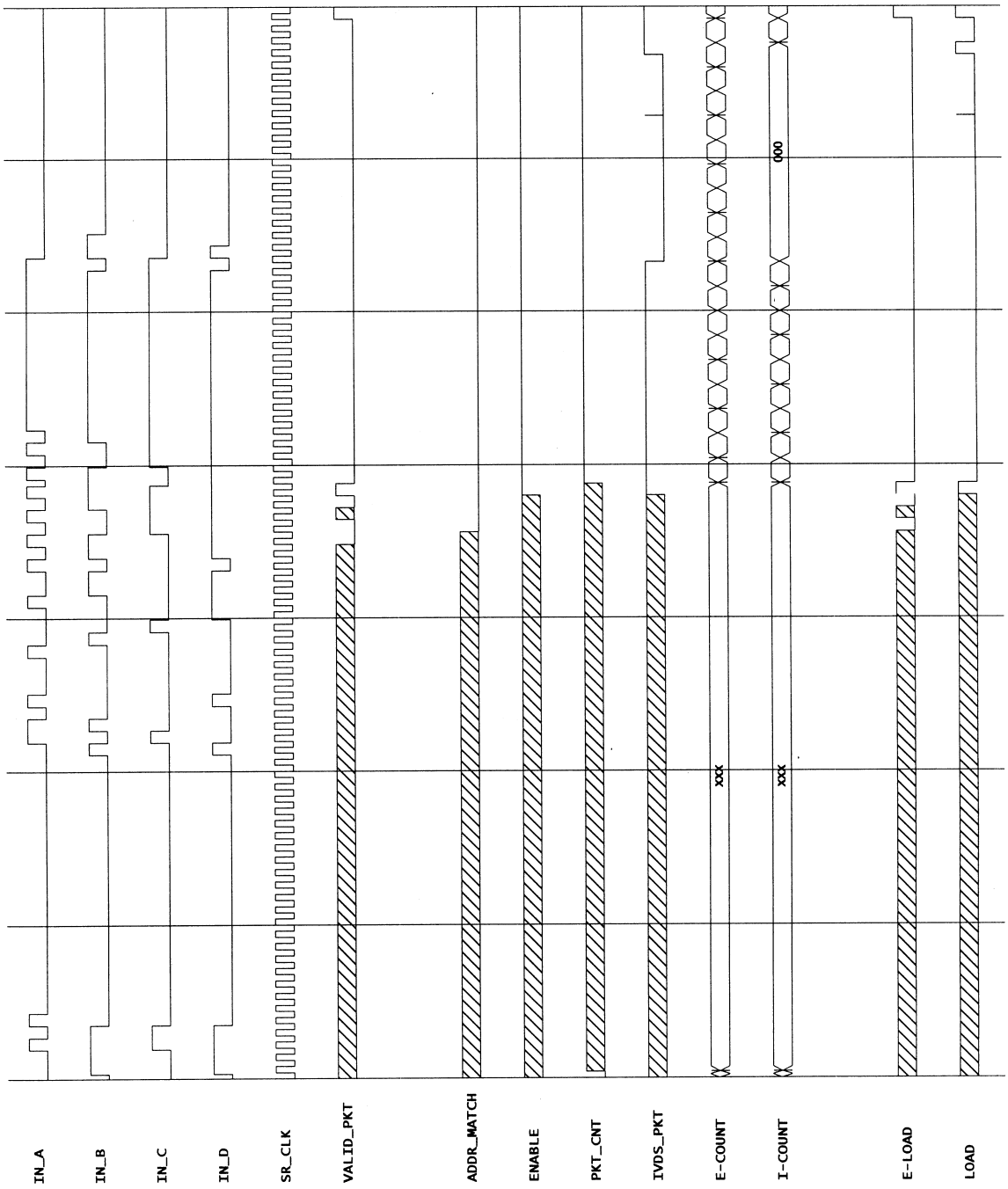

IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
I-COUNT
E-LOAD
LOAD
XXX
XXX

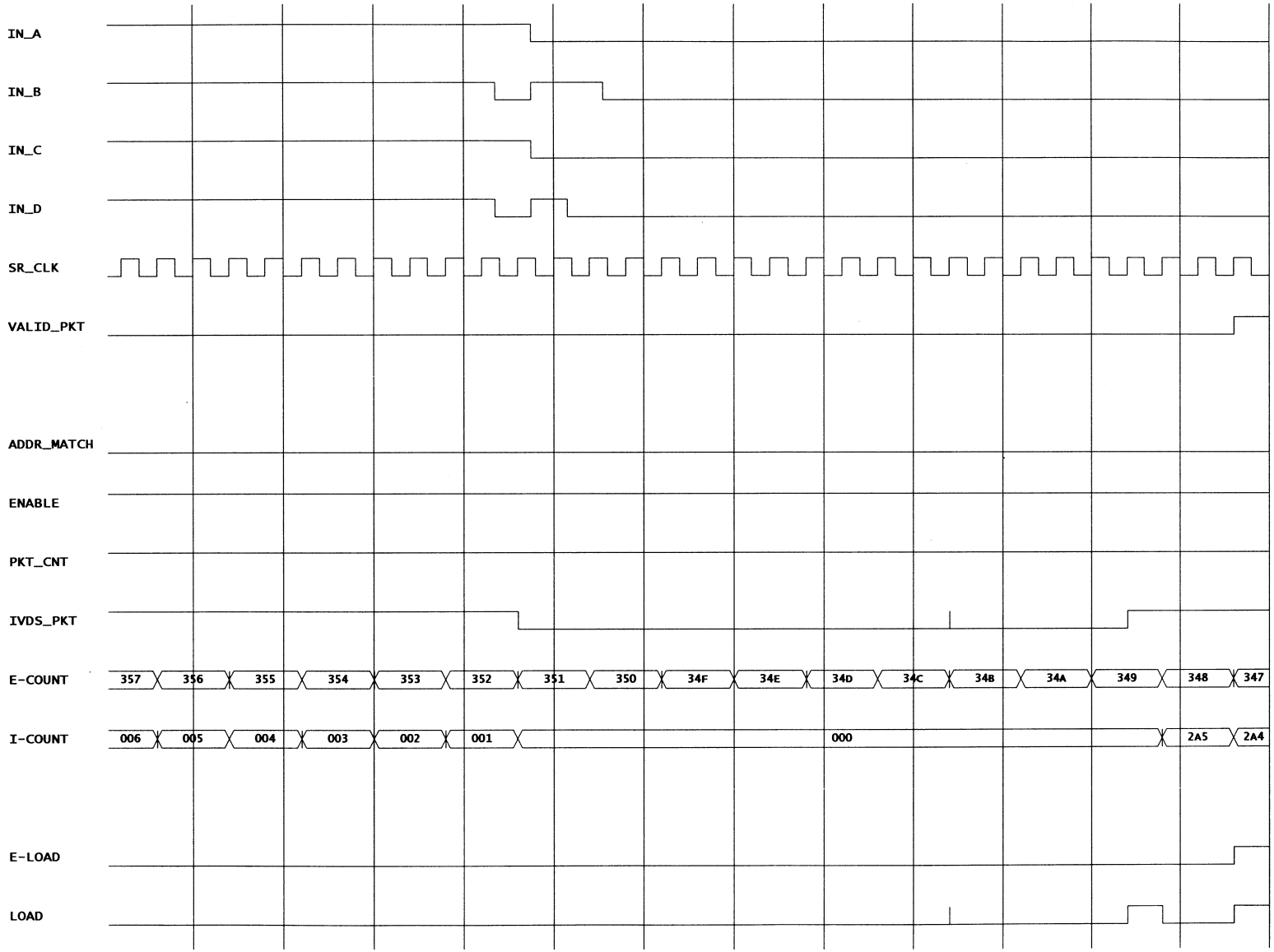
C.3 FPGA_TEST3

```
| this file shows the IVDS Length Counter properly load and count down to zero.
| the IVDS inter-packet counter counts down and properly loads the
| length value of the next IVDS packet.


net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector ivds_cnt ivds_cnt[7:0]
vector startaddr pac_in[23:16] pac_in[15:0]
vector inaddr inaddr[15:0]
vector pc_out pc_out[15:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector c c1 c0
vector a a1 a0
vector b b1 b0
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector control_cnt control_cnt3 control_cnt2 control_cnt1 control_cnt0
vector mode mode6 mode5 mode4
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]
vector q q3 q2 q1 q0
vector cc cc3 cc2 cc1 cc0

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0


wave fpga_test3.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count pkt  E-load load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1
```

```
| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8       9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 035A
wfm  in_a  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = A98421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000  palette = A
wfm in_a  @1615ns=0 @1655ns=1        | pallete bit 1  mode bit 1
wfm in_b  @1615ns=1 @1655ns=0                | pallete bit 2        mode bit 2
wfm in_c  @1615ns=0 @1655ns=0        | pallete bit 3        mode bit 3
wfm in_d  @1615ns=1 @1655ns=0        | pallete bit 4        not used
|                8         9                 format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1      | 20 16 12  8 4
0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1      | 21 17 13  9 5
1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 22 18 14 10 6
2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = ABCDEF
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 20 16 12  8 4
0
wfm in_b  @1935ns=1 @1955ns=1 @1995ns=0 @2035ns=0 @2075ns=1 @2115ns=1      | 21 17 13  9 5
1
wfm in_c  @1935ns=0 @1955ns=0 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 22 18 14 10 6
2
wfm in_d  @1935ns=1 @1955ns=1 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| 2155 2195 2235 2275 2315 2355 2395 2435 2475 2515 2555 2595 2635 2675 2715 2755 2795
2835 2875 2915 2955 2995

| IVDS packet Data bytes= 02A5
wfm  in_a  @2635ns=1 @2675ns=0 @2715ns=0 @2755ns=0    | --  8 4 0
wfm  in_b  @2635ns=0 @2675ns=1 @2715ns=1 @2755ns=0    | --  9 5 1
```

```
wfm  in_c  @2635ns=1 @2675ns=0 @2715ns=0 @2755ns=0    | -- 10 6 2
wfm  in_d  @2635ns=0 @2675ns=1 @2715ns=0 @2755ns=0    | -- -- 7 3
|                14       15        0        1       format
sim 3500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
I-COUNT
E-LOAD
LOAD
xxx
xxx
000

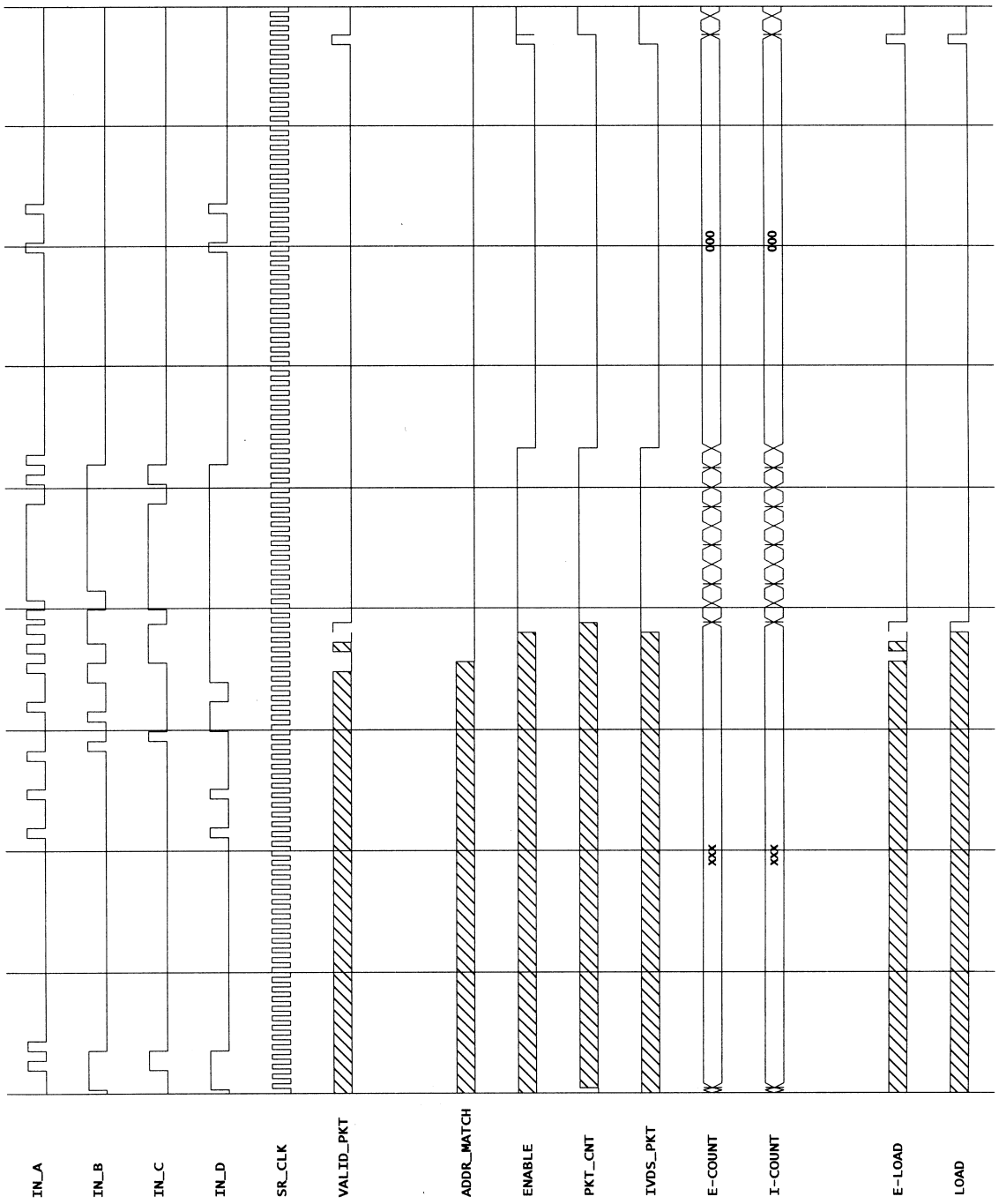

IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
357
356
355
354
353
352
351
350
34F
34E
34D
34C
34B
34A
349
348
347
I-COUNT
006
005
004
003
002
001
000
2A5
2A4
E-LOAD
LOAD

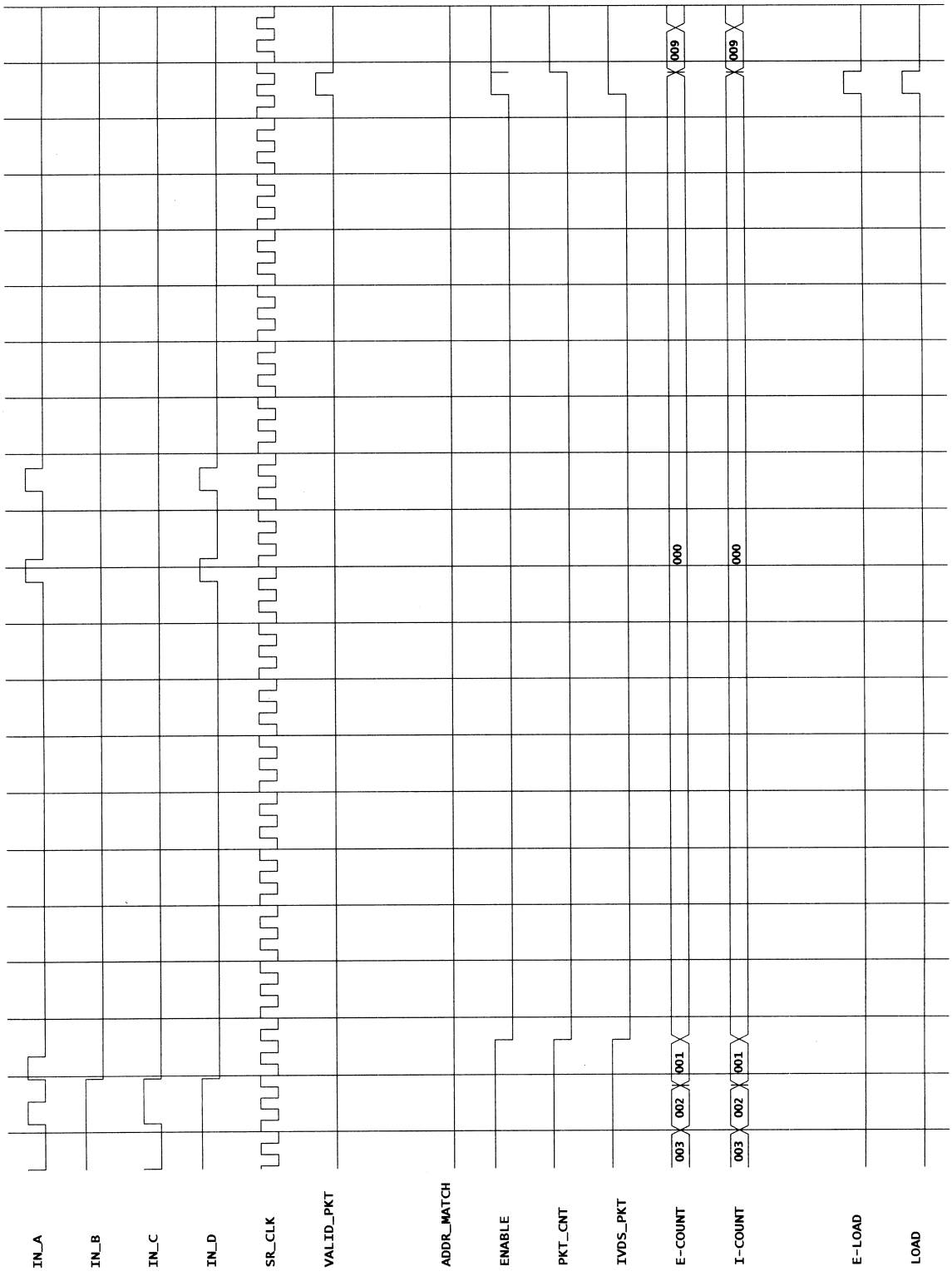
C.4 FPGA_TEST4

```
| this file shows a short Ethernet packet load, the counter count down,
| and the FPGA wait for a new Ethernet packet, detect it, and load it

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
| vector ivds_cnt ivds_cnt[7:0]
| vector startaddr pc_in[23:16] pc_in[15:0]
| vector inaddr inaddr[15:0]
| vector pc_out pc_out[15:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
| vector c c1 c0
| vector a a1 a0
| vector b b1 b0
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]


vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0


wave fpga_test4.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count pkt E-load load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
```

```
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8        9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 0009
wfm  in_a  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=0 @1095ns=0 @1135ns=0 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=0 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 0009
wfm  in_a  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000
wfm in_a  @1615ns=0 @1655ns=0         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=0                 | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                  format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1      | 20 16 12  8 4
0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1      | 21 17 13  9 5
1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 22 18 14 10 6
2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = ABCDEF
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=1 @2115ns=1      | 20 16 12  8 4
0
wfm in_b  @1935ns=1 @1955ns=1 @1995ns=0 @2035ns=0 @2075ns=1 @2115ns=1      | 21 17 13  9 5
1
wfm in_c  @1935ns=0 @1955ns=0 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 22 18 14 10 6
2
wfm in_d  @1935ns=1 @1955ns=1 @1995ns=1 @2035ns=1 @2075ns=1 @2115ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| 2155 2195 2235 2275 2315 2355 2395 2435 2475 2515 2555 2595 2635 2675 2715 2755 2795
2835 2875 2915 2955 2995

| Delimiter = AA
wfm  in_a  @2435ns=0 @2475ns=0
wfm  in_b  @2435ns=1 @2475ns=1
wfm  in_c  @2435ns=0 @2475ns=0
wfm  in_d  @2435ns=1 @2475ns=1
|               0       1
```

```
| Device Addr = 00001F
wfm  in_a  @2515ns=1 @2555ns=0 @2595ns=1 @2635ns=0 @2675ns=0 @2715ns=0 @2755ns=0 @2795ns=0
@2835ns=0 @2875ns=0 @2915ns=0 @2955ns=0
wfm  in_b  @2515ns=1 @2555ns=1 @2595ns=0 @2635ns=0 @2675ns=0 @2715ns=0 @2755ns=0 @2795ns=0
@2835ns=0 @2875ns=0 @2915ns=0 @2955ns=0
wfm  in_c  @2515ns=1 @2555ns=1 @2595ns=0 @2635ns=0 @2675ns=0 @2715ns=0 @2755ns=0 @2795ns=0
@2835ns=0 @2875ns=0 @2915ns=0 @2955ns=0
wfm  in_d  @2515ns=1 @2555ns=1 @2595ns=0 @2635ns=0 @2675ns=0 @2715ns=0 @2755ns=0 @2795ns=0
@2835ns=0 @2875ns=0 @2915ns=0 @2955ns=0
|               2        3       4        5        6        7        8        9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @2995ns=0   @3435ns=0
wfm  in_b  @2995ns=0   @3435ns=0
wfm  in_c  @2995ns=0   @3435ns=0
wfm  in_d  @2995ns=0   @3435ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @3475ns=1 @3515ns=0 @3555ns=0 @3595ns=0    | --  8 4 0
wfm  in_b  @3475ns=0 @3515ns=0 @3555ns=0 @3595ns=0    | --  9 5 1
wfm  in_c  @3475ns=0 @3515ns=0 @3555ns=0 @3595ns=0    | -- 10 6 2
wfm  in_d  @3475ns=1 @3515ns=0 @3555ns=0 @3595ns=0    | -- -- 7 3
|                  10        11        12        13      format

| IVDS packet Data bytes= 035A
wfm  in_a  @3635ns=1 @3675ns=0 @3715ns=0 @3755ns=0    | --  8 4 0
wfm  in_b  @3635ns=0 @3675ns=0 @3715ns=0 @3755ns=0    | --  9 5 1
wfm  in_c  @3635ns=0 @3675ns=0 @3715ns=0 @3755ns=0    | -- 10 6 2
wfm  in_d  @3635ns=1 @3675ns=0 @3715ns=0 @3755ns=0    | -- -- 7 3
|               14       15        0        1       format
sim 4500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
I-COUNT
E-LOAD
LOAD
000
000
xxx
xxx

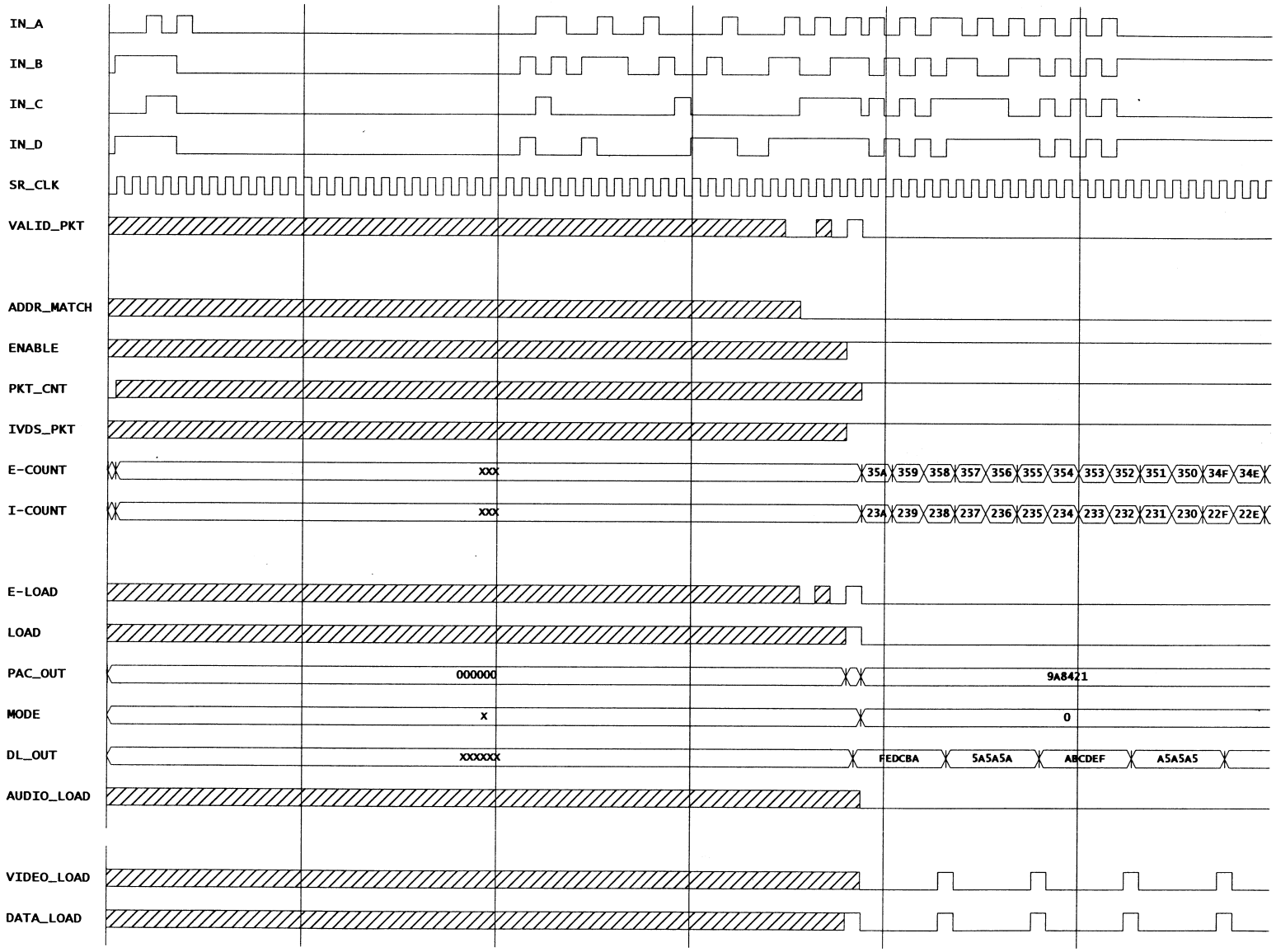

IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
I-COUNT
E-LOAD
LOAD
003
002
001
000
009

C.5 FPGA_TEST5

```
| this file test the basic ability to match the device addr of all ones
| and match the delimiter of 10101010
| this test file shifts data into a shift register
| and loads the output when the delimiter and addr match
| the counter loads with the desired value and begins counting.
| While it is counting the circuit will be enabled,
| when the counter reaches zero the Ethernet packet
| is over so the circuit should be disabled by enable going low.
| NOTE a byte counter needs to be used to enable the Length counter's clk.
| Also demonstrated are the 1 byte, 2byte, and 3 byte counters that are enabled
| by the ENABLE signal when the system receives a valid Ethernet packet

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0

wave fpga_test5.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count device E-load load pac_out mode dl_out audio_load pkt
video_load data_load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
```

```
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8       9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 023A
wfm  in_a  @1215ns=0 @1255ns=1 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=1 @1255ns=1 @1295ns=1 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000
wfm in_a  @1615ns=0 @1655ns=0         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=0                | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                 format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1      | 20 16 12  8 4
0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1      | 21 17 13  9 5
1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 22 18 14 10 6
2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = A5A5A5
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 20 16 12  8 4
0
wfm in_b  @1935ns=1 @1955ns=0 @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0      | 21 17 13  9 5
1
wfm in_c  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 22 18 14 10 6
2
wfm in_d  @1935ns=1 @1955ns=0 @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| 2155 2195 2235 2275 2315 2355 2395 2435 2475 2515 2555 2595 2635 2675 2715 2755 2795
2835 2875 2915 2955 2995

| IVDS DATA = FEDCBA
wfm in_a  @2155ns=1 @2195ns=0 @2235ns=1 @2275ns=0 @2315ns=1 @2355ns=0      | 20 16 12  8 4
0
```

```
wfm in_b  @2155ns=1 @2195ns=1 @2235ns=0 @2275ns=0 @2315ns=1 @2355ns=1      | 21 17 13  9 5
1
wfm in_c  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1 @2315ns=0 @2355ns=0      | 22 18 14 10 6
2
wfm in_d  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1 @2315ns=1 @2355ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = 5A5A5A
wfm in_a  @2395ns=1 @2435ns=0 @2475ns=1 @2515ns=0 @2555ns=1 @2595ns=0      | 20 16 12  8 4
0
wfm in_b  @2395ns=0 @2435ns=1 @2475ns=0 @2515ns=1 @2555ns=0 @2595ns=1      | 21 17 13  9 5
1
wfm in_c  @2395ns=1 @2435ns=0 @2475ns=1 @2515ns=0 @2555ns=1 @2595ns=0      | 22 18 14 10 6
2
wfm in_d  @2395ns=0 @2435ns=1 @2475ns=0 @2515ns=1 @2555ns=0 @2595ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format
sim 3500ns
```

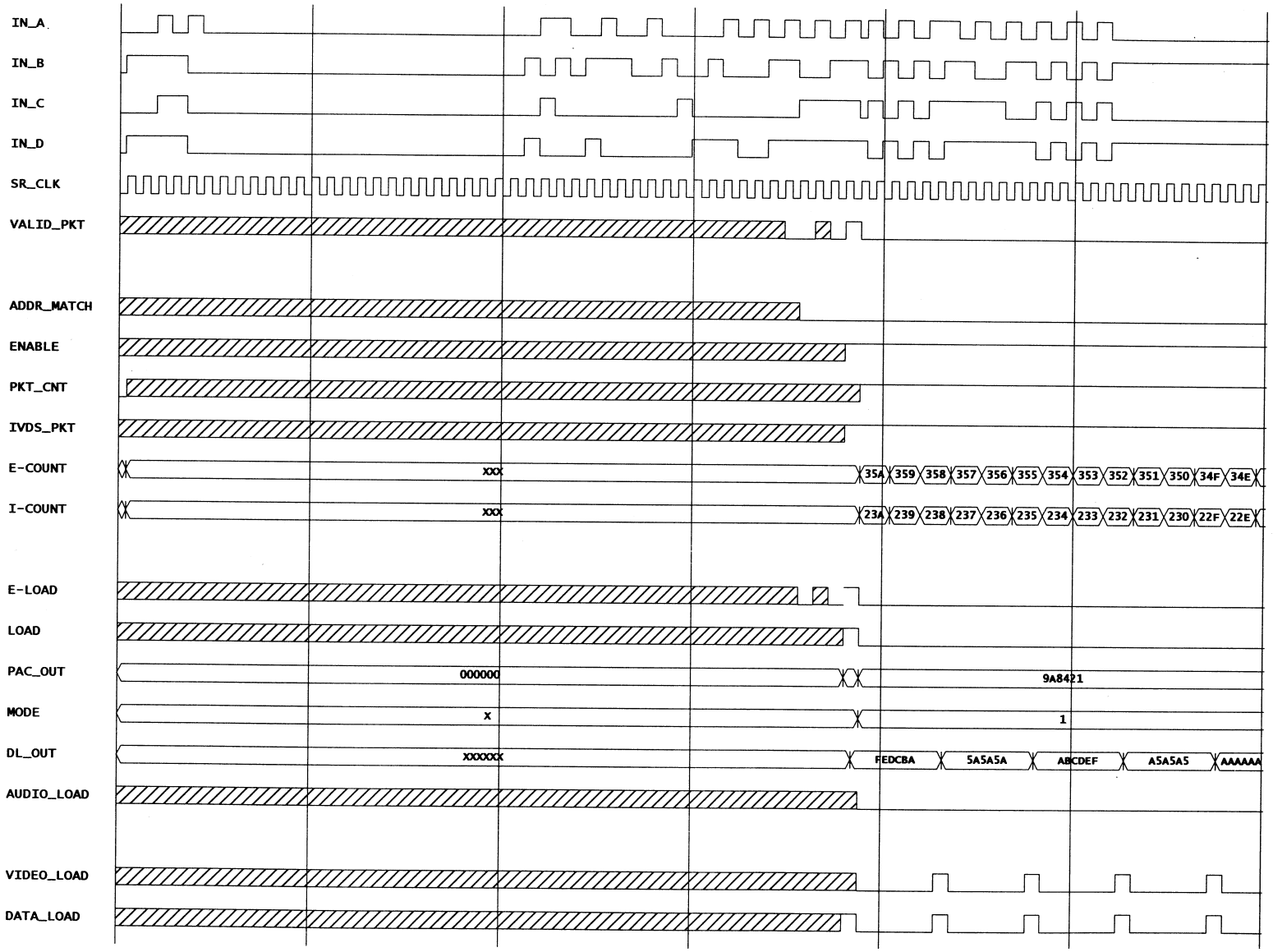

IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
xxx
35A 359 358 357 356 355 354 353 352 351 350 34F 34E
I-COUNT
xxx
23A 239 238 237 236 235 234 233 232 231 230 22F 22E
E-LOAD
LOAD
PAC_OUT
000000
9A8421
MODE
x
0
DL_OUT
xxxxxx
FEDCBA 5A5A5A ABCDEF A5A5A5
AUDIO_LOAD
VIDEO_LOAD
DATA_LOAD

C.6 FPGA_TEST6

```
| this file test the basic ability to match the device addr of all ones
| and match the delimiter of 10101010
| this test file shifts data into a shift register
| and loads the output when the delimiter and addr match
| the counter loads with the desired value and begins counting.
| While it is counting the circuit will be enabled,
| when the counter reaches zero the Ethernet packet
| is over so the circuit should be disabled by enable going low.
| NOTE a byte counter needs to be used to enable the Length counter's clk.
| Also demonstrated are the 1 byte, 2byte, and 3 byte counters that are enabled
| by the ENABLE signal when the system receives a valid Ethernet packet

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0


wave fpga_test6.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count device E-load load pac_out mode dl_out audio_load pkt
video_load data_load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
```

```
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8       9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 023A
wfm  in_a  @1215ns=0 @1255ns=1 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=1 @1255ns=1 @1295ns=1 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 001
wfm in_a  @1615ns=0 @1655ns=1         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=0                | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                 format

| IVDS DATA = ABCDEF
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1 @1855ns=0 @1895ns=1      | 20 16 12  8 4
0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0 @1855ns=1 @1895ns=1      | 21 17 13  9 5
1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 22 18 14 10 6
2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1 @1855ns=1 @1895ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = A5A5A5
wfm in_a  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 20 16 12  8 4
0
wfm in_b  @1935ns=1 @1955ns=0 @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0      | 21 17 13  9 5
1
wfm in_c  @1935ns=0 @1955ns=1 @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1      | 22 18 14 10 6
2
wfm in_d  @1935ns=1 @1955ns=0 @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| 2155 2195 2235 2275 2315 2355 2395 2435 2475 2515 2555 2595 2635 2675 2715 2755 2795
2835 2875 2915 2955 2995

| IVDS DATA = FEDCBA
wfm in_a  @2155ns=1 @2195ns=0 @2235ns=1 @2275ns=0 @2315ns=1 @2355ns=0      | 20 16 12  8 4
0
```

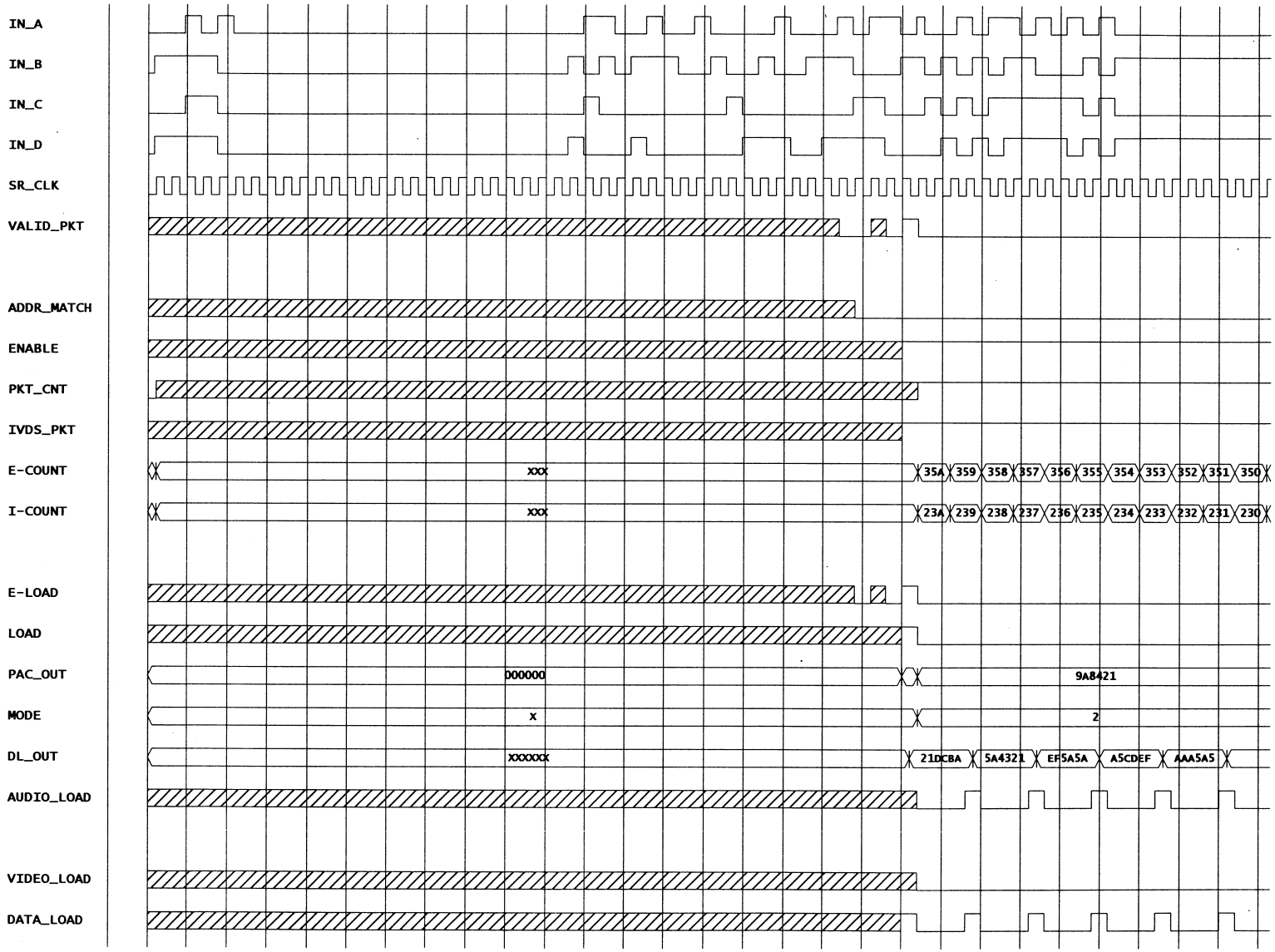
```
wfm in_b  @2155ns=1 @2195ns=1 @2235ns=0 @2275ns=0 @2315ns=1 @2355ns=1      | 21 17 13  9 5
1
wfm in_c  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1 @2315ns=0 @2355ns=0      | 22 18 14 10 6
2
wfm in_d  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1 @2315ns=1 @2355ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format

| IVDS DATA = 5A5A5A
wfm in_a  @2395ns=1 @2435ns=0 @2475ns=1 @2515ns=0 @2555ns=1 @2595ns=0      | 20 16 12  8 4
0
wfm in_b  @2395ns=0 @2435ns=1 @2475ns=0 @2515ns=1 @2555ns=0 @2595ns=1      | 21 17 13  9 5
1
wfm in_c  @2395ns=1 @2435ns=0 @2475ns=1 @2515ns=0 @2555ns=1 @2595ns=0      | 22 18 14 10 6
2
wfm in_d  @2395ns=0 @2435ns=1 @2475ns=0 @2515ns=1 @2555ns=0 @2595ns=1      | 23 19 15 11 7
3
|               10       11       12       13       14       15             format
sim 3500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
XXX
35A 359 358 357 356 355 354 353 352 351 350 34F 34E
I-COUNT
XXX
23A 239 238 237 236 235 234 233 232 231 230 22F 22E
E-LOAD
LOAD
PAC_OUT
000000
9A8421
MODE
X
1
DL_OUT
XXXXXX
FEDCBA 5A5A5A ABCDEF A5A5A5 AAAAAA
AUDIO_LOAD
VIDEO_LOAD
DATA_LOAD

C.7 FPGA_TEST7

```
| Ethernet packet loaded in broadcast mode
| PAC_OUT shows PAC loaded proper value
| MODE shows that packet data is mode 1 = NEW AUDIO mode
| DL_OUT shows proper values loaded every 160ns according to Data_load signal

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0

wave fpga_test7.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count device E-load load pac_out mode dl_out audio_load pkt
video_load data_load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
```

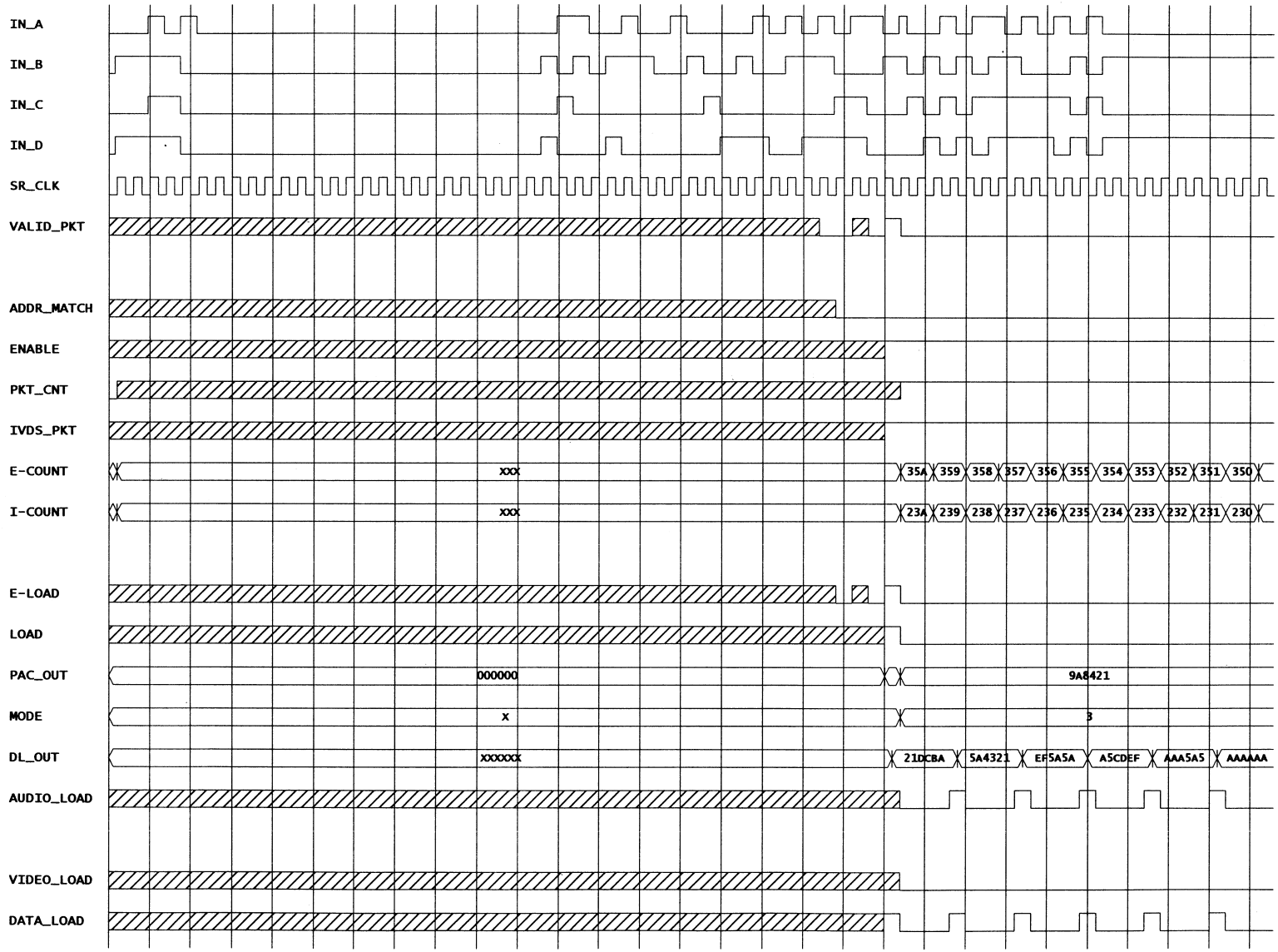
```
|               2        3       4        5        6        7        8        9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 023A
wfm  in_a  @1215ns=0 @1255ns=1 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=1 @1255ns=1 @1295ns=1 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|                14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 000
wfm in_a  @1615ns=0 @1655ns=0         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=1                | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                 format

| IVDS DATA = ABCD
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1    | 12  8 4 0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0    | 13  9 5 1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1    | 14 10 6 2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 1234
wfm in_a  @1855ns=1 @1895ns=0 @1935ns=1 @1955ns=0    | 12  8 4 0
wfm in_b  @1855ns=0 @1895ns=1 @1935ns=1 @1955ns=0    | 13  9 5 1
wfm in_c  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=1    | 14 10 6 2
wfm in_d  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=0    | 15 11 7 3
|            14       15        10       11           format

| IVDS DATA = A5A5
wfm in_a   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 12  8 4 0
wfm in_b   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 13  9 5 1
wfm in_c   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 14 10 6 2
wfm in_d   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 15 11 7 3
|                 12       13       14       15      format


| IVDS DATA = FEDC
wfm in_a  @2155ns=1 @2195ns=0 @2235ns=1 @2275ns=0    | 12  8 4 0
wfm in_b  @2155ns=1 @2195ns=1 @2235ns=0 @2275ns=0    | 13  9 5 1
wfm in_c  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 14 10 6 2
wfm in_d  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 5A5A
wfm in_a  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 12  8 4 0
wfm in_b  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 13  9 5 1
wfm in_c  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 14 10 6 2
wfm in_d  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 15 11 7 3
|               10       11       12       13         format
sim 3500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
XXX
35A 359 358 357 356 355 354 353 352 351 350
I-COUNT
XXX
23A 239 238 237 236 235 234 233 232 231 230
E-LOAD
LOAD
PAC_OUT
000000
9A8421
MODE
X
2
DL_OUT
XXXXXX
21DCBA 5A4321 EF5A5A A5CDEF AAA5A5
AUDIO_LOAD
VIDEO_LOAD
DATA_LOAD

C.8 FPGA_TEST8

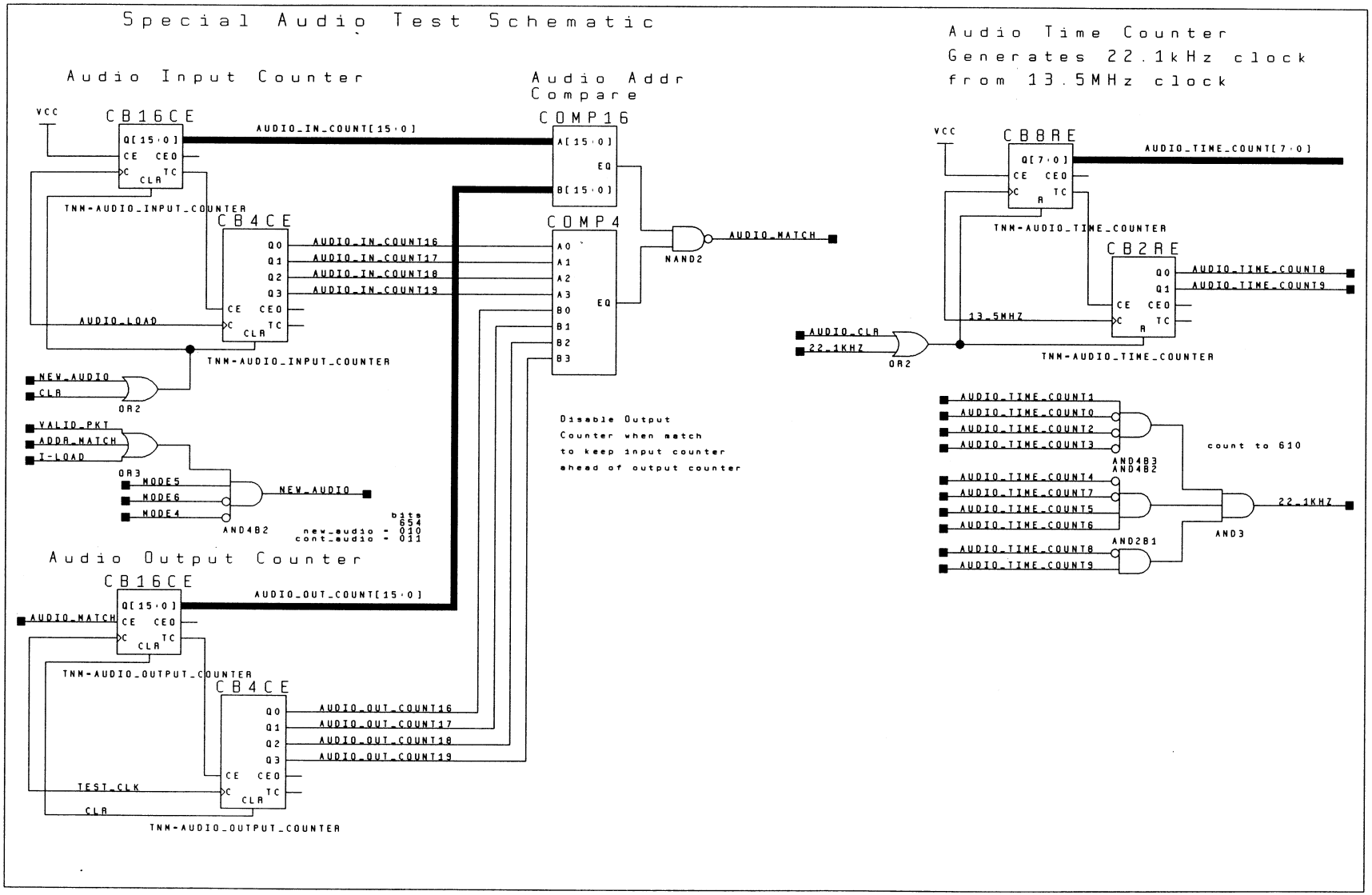
```
| Ethernet packet loaded in broadcast mode
| PAC_OUT shows PAC loaded proper value
| MODE shows that packet data is mode 1 = NEW AUDIO mode
| DL_OUT shows proper values loaded every 160ns according to Data_load signal

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0

wave fpga_test8.wfm in_a in_b in_c in_d sr_clk valid_pkt pkt_head addr_match enable
pkt_cnt ivds_pkt E-count I-count device E-load load pac_out mode dl_out audio_load pkt
video_load data_load

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1

| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
```

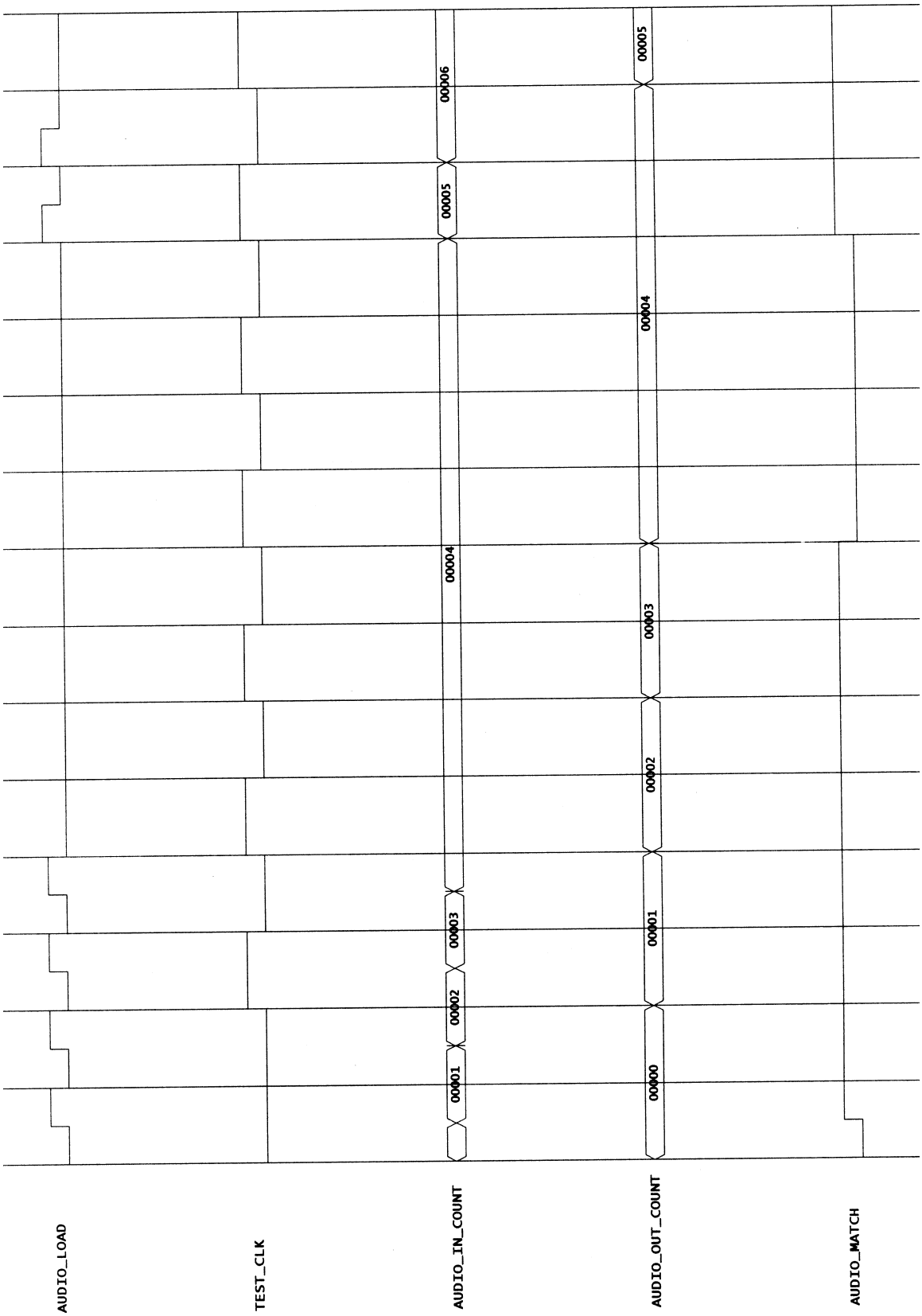
```
|               2        3       4        5        6        7        8        9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 023A
wfm  in_a  @1215ns=0 @1255ns=1 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=1 @1255ns=1 @1295ns=1 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|                14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 011
wfm in_a  @1615ns=0 @1655ns=1         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=1                | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                 format

| IVDS DATA = ABCD
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1    | 12  8 4 0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0    | 13  9 5 1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1    | 14 10 6 2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 1234
wfm in_a  @1855ns=1 @1895ns=0 @1935ns=1 @1955ns=0    | 12  8 4 0
wfm in_b  @1855ns=0 @1895ns=1 @1935ns=1 @1955ns=0    | 13  9 5 1
wfm in_c  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=1    | 14 10 6 2
wfm in_d  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=0    | 15 11 7 3
|            14       15        10       11           format

| IVDS DATA = A5A5
wfm in_a   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 12  8 4 0
wfm in_b   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 13  9 5 1
wfm in_c   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 14 10 6 2
wfm in_d   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 15 11 7 3
|                 12       13       14       15      format


| IVDS DATA = FEDC
wfm in_a  @2155ns=1 @2195ns=0 @2235ns=1 @2275ns=0    | 12  8 4 0
wfm in_b  @2155ns=1 @2195ns=1 @2235ns=0 @2275ns=0    | 13  9 5 1
wfm in_c  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 14 10 6 2
wfm in_d  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 5A5A
wfm in_a  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 12  8 4 0
wfm in_b  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 13  9 5 1
wfm in_c  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 14 10 6 2
wfm in_d  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 15 11 7 3
|               10       11       12       13         format
sim 3500ns
```


IN_A
IN_B
IN_C
IN_D
SR_CLK
VALID_PKT
ADDR_MATCH
ENABLE
PKT_CNT
IVDS_PKT
E-COUNT
xxx
35A 359 358 357 356 355 354 353 352 351 350
I-COUNT
xxx
23A 239 238 237 236 235 234 233 232 231 230
E-LOAD
LOAD
PAC_OUT
000000
9A8421
MODE
x
3
DL_OUT
xxxxxx
21DCBA 5A4321 EF5A5A A5CDEF AAA5A5 AAAAAA
AUDIO_LOAD
VIDEO_LOAD
DATA_LOAD

C.9 FPGA_TEST9

```
| uses special schematic "audio_test"
| The Audio Input Counter is allowed to count up to 3 then hold at 3
| The Audio Output Counter counts upto 4 and the AUDIO_MATCH signal goes low
| indicating that a match has been made.  The TEST_CLK signal continues but until
| the AUDIO_LOAD clock allows the Audio Input Counter to increment the
| Audio Output Counter can not increment.


net audio_test.vsm

restart

stepsize 10.0ns

vector audio_out_count audio_out_count19 audio_out_count18 audio_out_count17
audio_out_count16 audio_out_count[15:0]
vector audio_in_count audio_in_count19 audio_in_count18 audio_in_count17 audio_in_count16
audio_in_count[15:0]

wave fpga_test9.wfm  audio_load test_clk audio_in_count audio_out_count audio_match

l mode4 mode5 mode6 i-load valid_pkt
wfm clr @0ns=1 @15ns=0

wfm  audio_load  @0ns=0 @50ns=1 @100ns=0 @150ns=1 @200ns=0 @250ns=1 @300ns=0 @350ns=1
@400ns=0 @1200ns=1 @1250ns=0 @1300ns=1 @1350ns=0
wfm  test_clk  @0ns=0 @200ns=1 @300ns=0 @400ns=1 @500ns=0 @600ns=1 @700ns=0 @800ns=1
@900ns=0 @1000ns=1 @1100ns=0 @1200ns=1 @1300ns=0 @1400ns=1

sim 1500ns
```


Special Audio Test Schematic
Audio Input Counter
VCC
CB16CE
Q[15:0]
CE CEO
C TC
CLR
AUDIO_IN_COUNT[15:0]
TNM-AUDIO_INPUT_COUNTER
CB4CE
Q0
Q1
Q2
Q3
AUDIO_IN_COUNT16
AUDIO_IN_COUNT17
AUDIO_IN_COUNT18
AUDIO_IN_COUNT19
AUDIO_LOAD
TNM-AUDIO_INPUT_COUNTER
NEW_AUDIO
CLR
OR2
VALID_PKT
ADDR_MATCH
I-LOAD
OR3
MODE5
MODE6
MODE4
AND4B2
NEW_AUDIO
bits
654
new_audio - 010
cont_audio - 011
Audio Output Counter
CB16CE
Q[15:0]
AUDIO_MATCH
CE CEO
C TC
CLR
AUDIO_OUT_COUNT[15:0]
TNM-AUDIO_OUTPUT_COUNTER
CB4CE
Q0
Q1
Q2
Q3
AUDIO_OUT_COUNT16
AUDIO_OUT_COUNT17
AUDIO_OUT_COUNT18
AUDIO_OUT_COUNT19
TEST_CLK
CLR
TNM-AUDIO_OUTPUT_COUNTER
Audio Addr Compare
COMP16
A[15:0]
EQ
B[15:0]
COMP4
A0
A1
A2
A3
B0
B1
B2
B3
EQ
NAND2
AUDIO_MATCH
Disable Output
Counter when match
to keep input counter
ahead of output counter
Audio Time Counter
Generates 22.1kHz clock
from 13.5MHz clock
VCC
CB8RE
Q[7:0]
CE CEO
C TC
R
AUDIO_TIME_COUNT[7:0]
TNM-AUDIO_TIME_COUNTER
CB2RE
Q0
Q1
AUDIO_TIME_COUNT8
AUDIO_TIME_COUNT9
CE CEO
C TC
R
13.5MHZ
TNM-AUDIO_TIME_COUNTER
AUDIO_CLR
22.1KHZ
OR2
AUDIO_TIME_COUNT1
AUDIO_TIME_COUNT0
AUDIO_TIME_COUNT2
AUDIO_TIME_COUNT3
AND4B3
AND4B2
count to 610
AUDIO_TIME_COUNT4
AUDIO_TIME_COUNT7
AUDIO_TIME_COUNT5
AUDIO_TIME_COUNT6
AND2B1
AUDIO_TIME_COUNT8
AUDIO_TIME_COUNT9
AND3
22.1KHZ

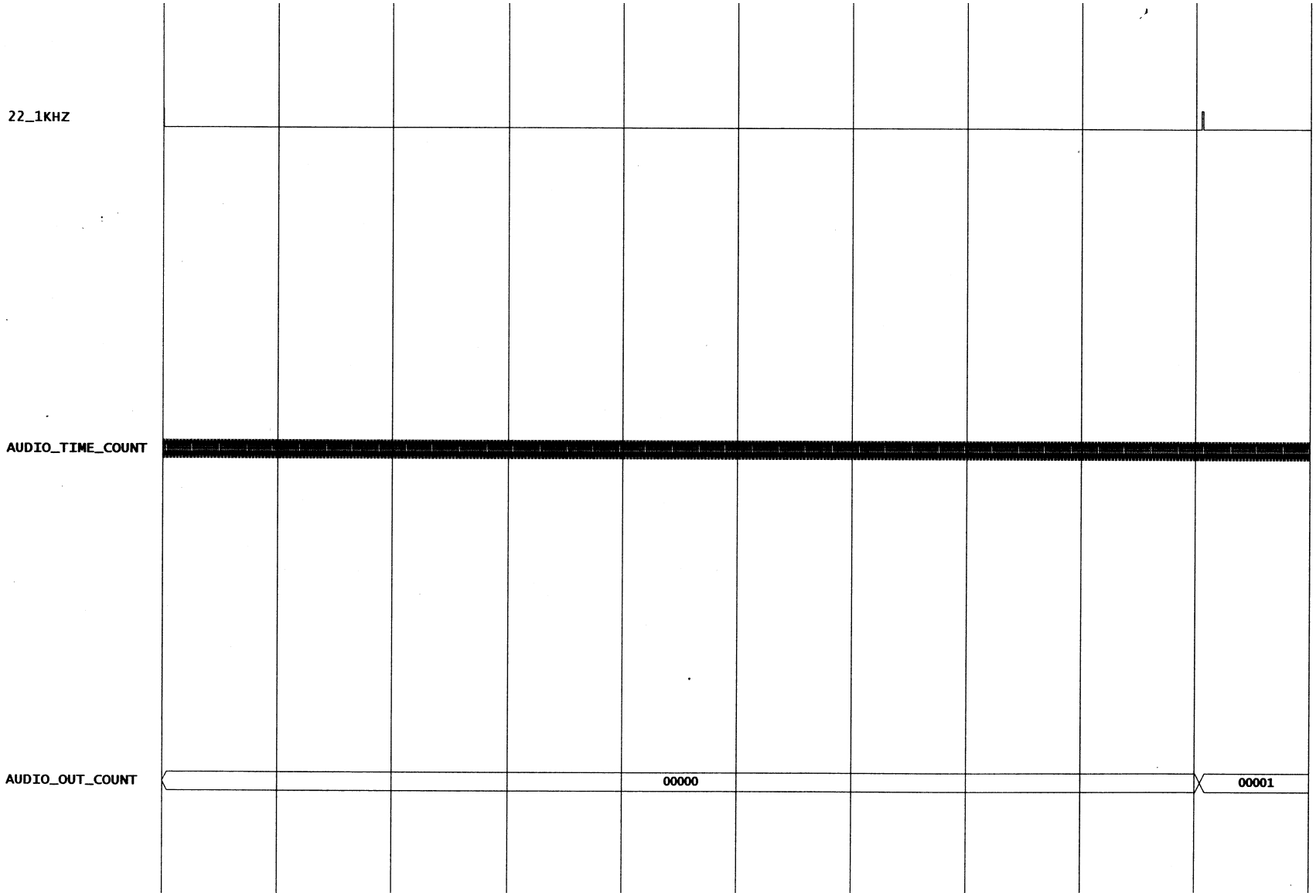

AUDIO_LOAD
TEST_CLK
AUDIO_IN_COUNT
00001
00002
00003
00004
00005
00006
AUDIO_OUT_COUNT
00000
00001
00002
00003
00004
00005
AUDIO_MATCH

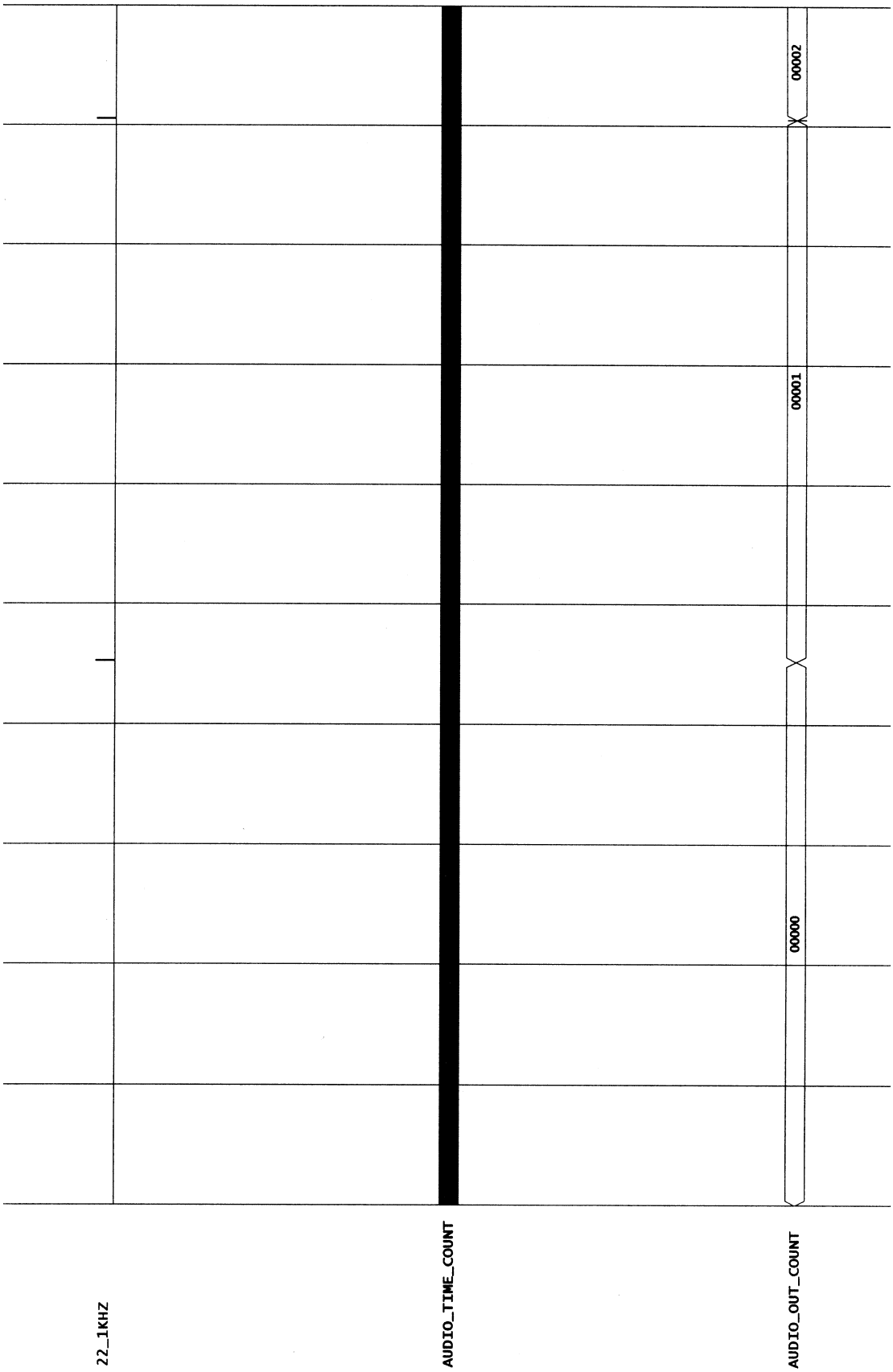
C.10 FPGA_TEST10

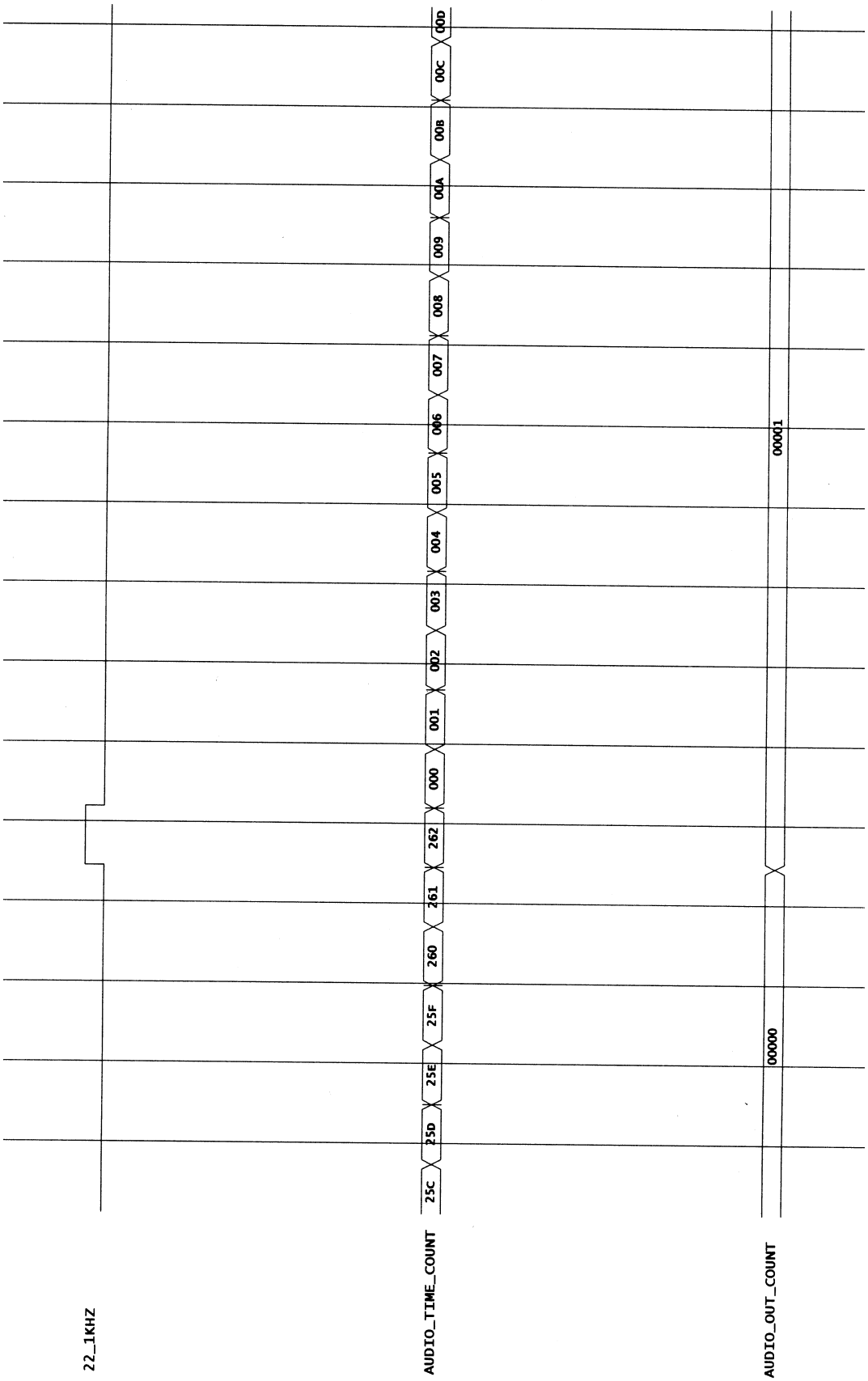
```
| Ethernet packet loaded in broadcast mode
| this file shows the Audio Time Counter incrementing using a 13.5MHz clock
| When the counter reaches 262 (hex) a series of and gates trigger generate
| a logic high on the 22_1KHZ line for 74.074ns and resets the counter.
| The 22_1KHZ signal is used as a clock for the Audio Output Counter whose
| output is shown as well.

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector audio_out_count audio_out_count19 audio_out_count18 audio_out_count17
audio_out_count16 audio_out_count[15:0]
vector audio_in_count audio_in_count19 audio_in_count18 audio_in_count17 audio_in_count16
audio_in_count[15:0]
vector audio_time_count audio_time_count9 audio_time_count8 audio_time_count[7:0]

vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0


wave fpga_test10.wfm 22_1khz audio_time_count audio_out_count

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )
after 7ns do(every 74.074ns do (wfm 13_5mhz @0ns=0 @37.037ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset
wfm pwrup @0ns=1 @35ns=0
wfm audio_clr @0ns=1 @80ns=0
wfm clr @0ns=1 @15ns=0


| init to zeros
wfm  in_a  @1ns=0   | lsb
wfm  in_b  @1ns=0
wfm  in_c  @1ns=0
wfm  in_d  @1ns=0   | msb

| Delimiter = AA
wfm  in_a  @15ns=0 @55ns=0
wfm  in_b  @15ns=1 @55ns=1
wfm  in_c  @15ns=0 @55ns=0
wfm  in_d  @15ns=1 @55ns=1
|               0       1
```

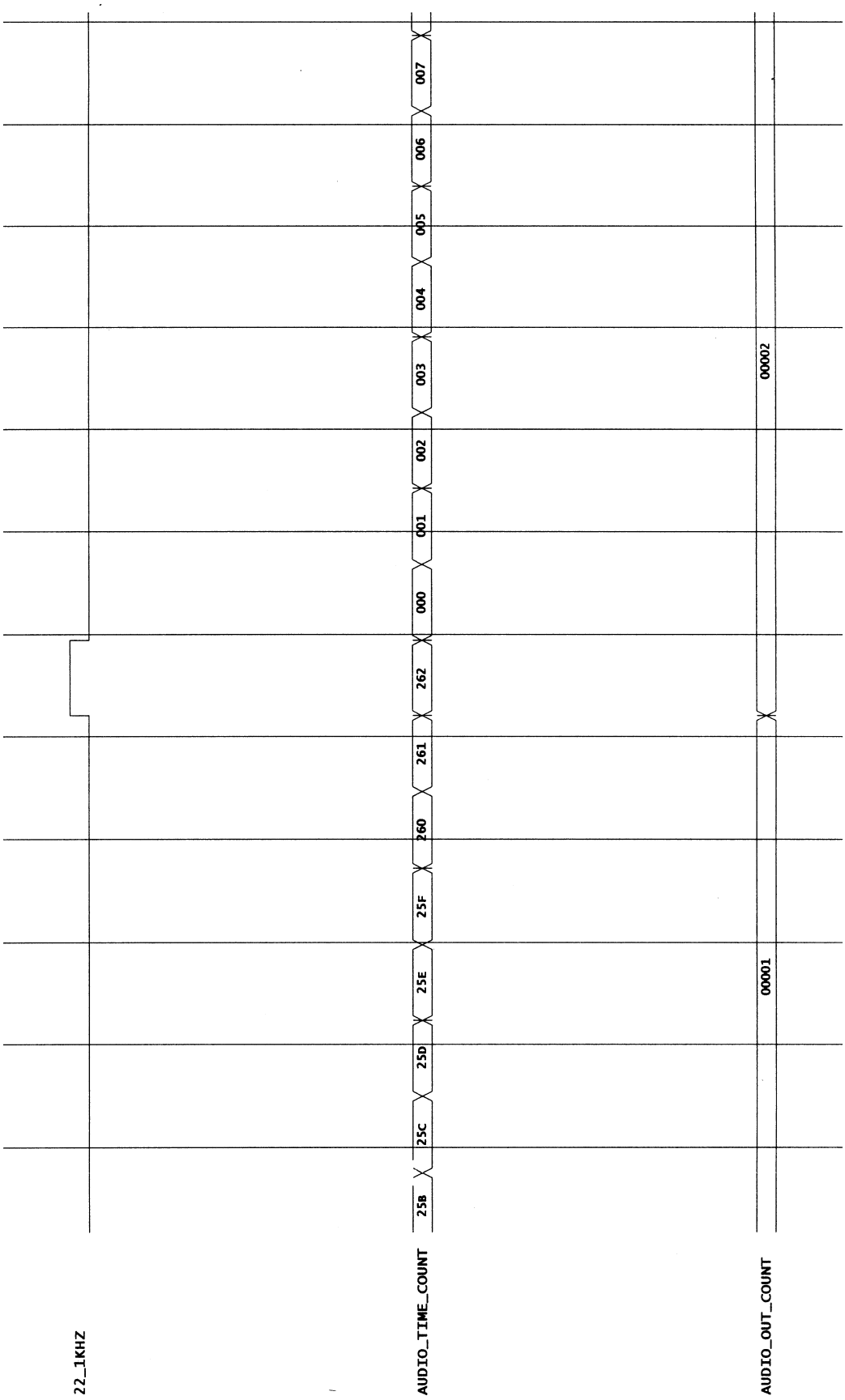
```
| Device Addr = 00001F
wfm  in_a  @95ns=1 @135ns=0 @175ns=1 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_b  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_c  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
wfm  in_d  @95ns=1 @135ns=1 @175ns=0 @215ns=0 @255ns=0 @295ns=0 @335ns=0 @375ns=0 @415ns=0
@455ns=0 @495ns=0 @535ns=0
|               2        3       4        5        6        7        8       9        10
11        12       13

| ignore Source Addr = 000000
wfm  in_a  @575ns=0   @1015ns=0
wfm  in_b  @575ns=0   @1015ns=0
wfm  in_c  @575ns=0   @1015ns=0
wfm  in_d  @575ns=0   @1015ns=0
|                14...         9

| Ethernet Length in bytes = 035A
wfm  in_a  @1055ns=0 @1095ns=1 @1135ns=1 @1175ns=0    | --  8 4 0
wfm  in_b  @1055ns=1 @1095ns=0 @1135ns=1 @1175ns=0    | --  9 5 1
wfm  in_c  @1055ns=0 @1095ns=1 @1135ns=0 @1175ns=0    | -- 10 6 2
wfm  in_d  @1055ns=1 @1095ns=0 @1135ns=0 @1175ns=0    | -- -- 7 3
|                 10        11        12        13      format

| IVDS packet Data bytes= 023A
wfm  in_a  @1215ns=0 @1255ns=1 @1295ns=0 @1335ns=0    | --  8 4 0
wfm  in_b  @1215ns=1 @1255ns=1 @1295ns=1 @1335ns=0    | --  9 5 1
wfm  in_c  @1215ns=0 @1255ns=0 @1295ns=0 @1335ns=0    | -- 10 6 2
wfm  in_d  @1215ns=1 @1255ns=0 @1295ns=0 @1335ns=0    | -- -- 7 3
|               14       15        0        1       format

| IVDS Starting Address = 9A8421
wfm  in_a  @1375ns=1 @1415ns=0 @1455ns=0 @1495ns=0 @1535ns=0 @1575ns=1
wfm  in_b  @1375ns=0 @1415ns=1 @1455ns=0 @1495ns=0 @1535ns=1 @1575ns=0
wfm  in_c  @1375ns=0 @1415ns=0 @1455ns=1 @1495ns=0 @1535ns=0 @1575ns=0
wfm  in_d  @1375ns=0 @1415ns=0 @1455ns=0 @1495ns=1 @1535ns=1 @1575ns=1
|                 2         3         4         5         6         7

| IVDS MODE = 011
wfm in_a  @1615ns=0 @1655ns=0         | pallete bit 1  mode bit 1
wfm in_b  @1615ns=0 @1655ns=1                | pallete bit 2         mode bit 2
wfm in_c  @1615ns=0 @1655ns=0         | pallete bit 3         mode bit 3
wfm in_d  @1615ns=0 @1655ns=0         | pallete bit 4         not used
|                8         9                 format

| IVDS DATA = ABCD
wfm in_a  @1695ns=0 @1735ns=1 @1775ns=0 @1815ns=1    | 12  8 4 0
wfm in_b  @1695ns=1 @1735ns=1 @1775ns=0 @1815ns=0    | 13  9 5 1
wfm in_c  @1695ns=0 @1735ns=0 @1775ns=1 @1815ns=1    | 14 10 6 2
wfm in_d  @1695ns=1 @1735ns=1 @1775ns=1 @1815ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 1234
wfm in_a  @1855ns=1 @1895ns=0 @1935ns=1 @1955ns=0    | 12  8 4 0
wfm in_b  @1855ns=0 @1895ns=1 @1935ns=1 @1955ns=0    | 13  9 5 1
wfm in_c  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=1    | 14 10 6 2
wfm in_d  @1855ns=0 @1895ns=0 @1935ns=0 @1955ns=0    | 15 11 7 3
|            14       15        10       11           format

| IVDS DATA = A5A5
wfm in_a   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 12  8 4 0
wfm in_b   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 13  9 5 1
wfm in_c   @1995ns=0 @2035ns=1 @2075ns=0 @2115ns=1   | 14 10 6 2
wfm in_d   @1995ns=1 @2035ns=0 @2075ns=1 @2115ns=0   | 15 11 7 3
|                 12       13       14       15      format

| 2155 2195 2235 2275 2315 2355 2395 2435 2475 2515 2555 2595 2635 2675 2715 2755 2795
2835 2875 2915 2955 2995

| IVDS DATA = FEDC
wfm in_a  @2155ns=1 @2195ns=0 @2235ns=1 @2275ns=0    | 12  8 4 0
wfm in_b  @2155ns=1 @2195ns=1 @2235ns=0 @2275ns=0    | 13  9 5 1
wfm in_c  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 14 10 6 2
```

```
wfm in_d  @2155ns=1 @2195ns=1 @2235ns=1 @2275ns=1    | 15 11 7 3
|               10       11       12       13         format

| IVDS DATA = 5A5A
wfm in_a  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 12  8 4 0
wfm in_b  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 13  9 5 1
wfm in_c  @2315ns=1 @2355ns=0 @2395ns=1 @2435ns=0    | 14 10 6 2
wfm in_d  @2315ns=0 @2355ns=1 @2395ns=0 @2435ns=1    | 15 11 7 3
|               10       11       12       13         format
sim 100000ns
```


22_1KHZ
AUDIO_TIME_COUNT
AUDIO_OUT_COUNT
00000
00001

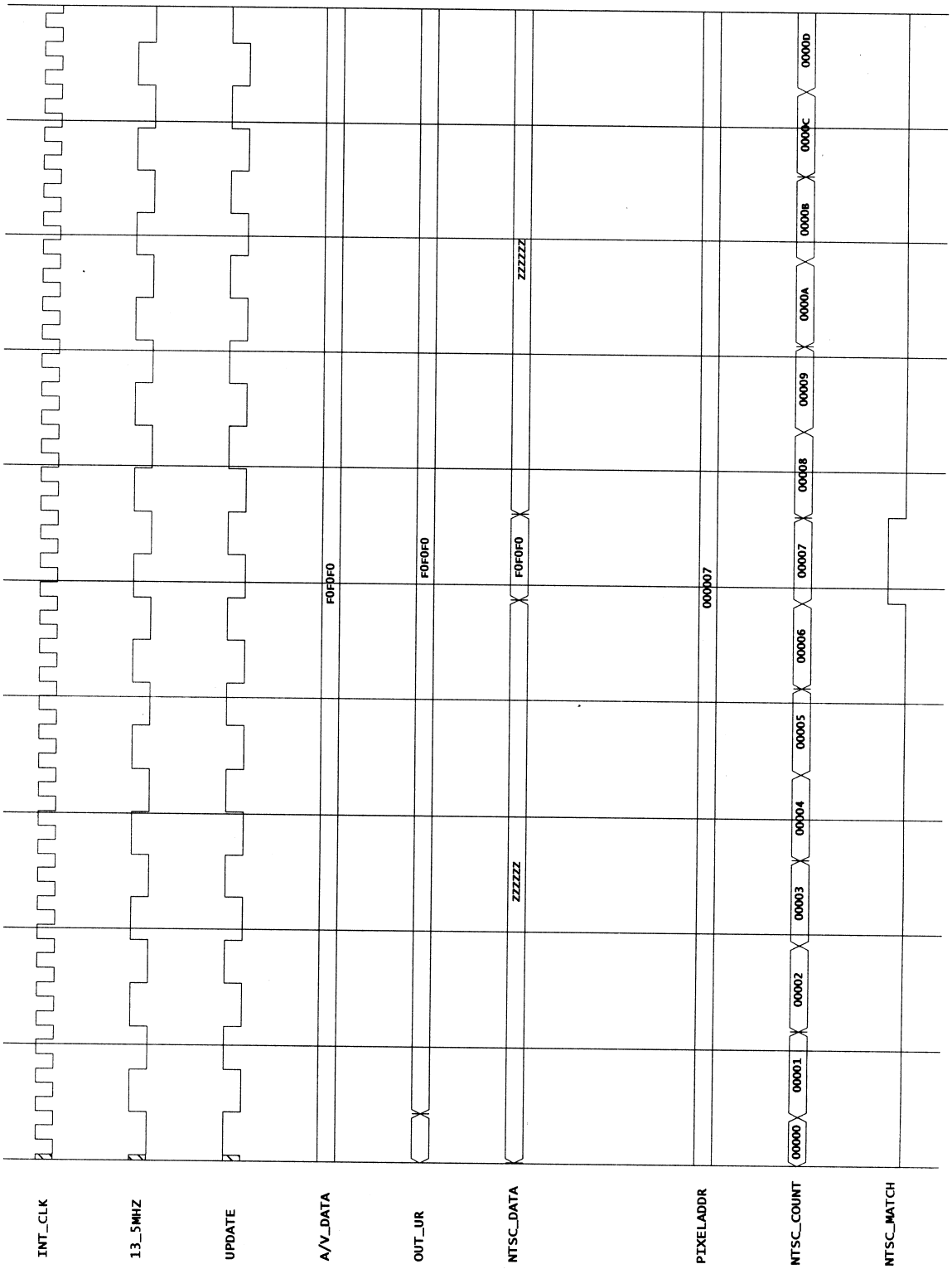

22_1KHZ
AUDIO_TIME_COUNT
AUDIO_OUT_COUNT
00000
00001
00002


22_1KHZ
AUDIO_TIME_COUNT
25C
25D
25E
25F
260
261
262
000
001
002
003
004
005
006
007
008
009
00A
00B
00C
00D
AUDIO_OUT_COUNT
00000
00001


22_1KHZ
AUDIO_TIME_COUNT
25B
25C
25D
25E
25F
260
261
262
000
001
002
003
004
005
006
007
AUDIO_OUT_COUNT
00001
00002

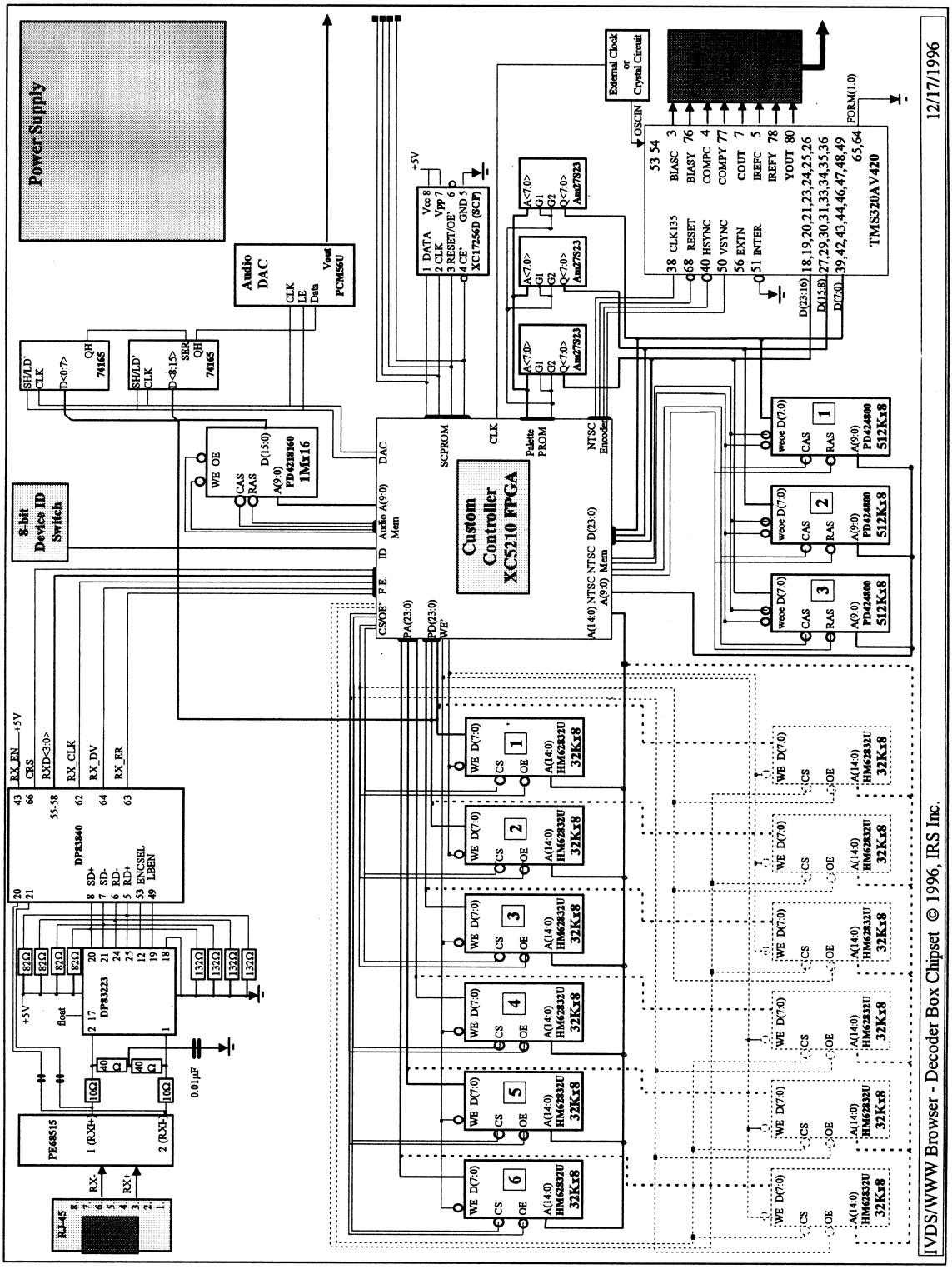
C.11 FPGA_TEST11

```
| Ethernet packet loaded in broadcast mode
| this file shows the Audio Time Counter incrementing using a 13.5MHz clock
| When the counter reaches 262 (hex) a series of and gates trigger generate
| a logic high on the 22_1KHZ line for 74.074ns and resets the counter.
| The 22_1KHZ signal is used as a clock for the Audio Output Counter whose
| output is shown as well.

net fpga_v4.vsm

restart

stepsize 10.0ns

vector data_a data_a[0:15]
vector data_b data_b[0:15]
vector data_c data_c[0:15]
vector data_d data_d[0:15]
vector device device[7:0]
vector E-count count10 count9 count8 count[7:0]
vector I-count icount10 icount9 icount8 icount[7:0]
vector pac_out pac_out[23:16] pac_out[15:0]
vector pixeladdr pixeladdr[23:16] pixeladdr[15:0]
vector oc_out oc_out[15:0]
vector ic_out ic_out[15:0]
vector mode mode6 mode5 mode4
vector dl_out out_dr[23:16] out_dr[15:8] out_dr[7:0]
vector audio_out_count audio_out_count19 audio_out_count18 audio_out_count17
audio_out_count16 audio_out_count[15:0]
vector audio_in_count audio_in_count19 audio_in_count18 audio_in_count17 audio_in_count16
audio_in_count[15:0]
vector audio_time_count audio_time_count9 audio_time_count8 audio_time_count[7:0]

vector a/v_data a/v_data[23:16] a/v_data[15:8] a/v_data[7:0]
vector databus1 databus1[15:0] databus1[23:16]
vector databus2 databus1[15:0] databus1[23:16]
vector addrbus1 addrbus1[15:0]
vector out_ur out_ur[23:16] out_ur[15:8] out_ur[7:0]
vector ntsc_data ntsc_data[23:16] ntsc_data[15:8] ntsc_data[7:0]
vector ntsc_count ntsc_count19 ntsc_count18 ntsc_count17 ntsc_count16 ntsc_count[15:0]
vector ntsc_addr ntsc_addr19 ntsc_addr18 ntsc_addr17 ntsc_addr16 ntsc_addr[15:0]

vector ivds_data  data_d10 data_c10 data_b10 data_a10 data_d11 data_c11 data_b11 data_a11
data_d12 data_c12 data_b12 data_a12 data_d13 data_c13 data_b13 data_a13 data_d14 data_c14
data_b14 data_a14 data_d15 data_c15 data_b15 data_a15

vector out_dr  out_dr23 out_dr22 out_dr21 out_dr20 out_dr19 out_dr18 out_dr17 out_dr16
out_dr15 out_dr14 out_dr13 out_dr12 out_dr11 out_dr10 out_dr9 out_dr8 out_dr7 out_dr6
out_dr5 out_dr4 out_dr3 out_dr2 out_dr1 out_dr0

wave fpga_test11.wfm int_clk 13_5mhz update a/v_data out_ur ntsc_data sr_r pixeladdr
ntsc_count ntsc_match

every 40ns do (wfm sr_clk @0ns=0 @20ns=1)
after 5ns do(every 24.69ns do (wfm int_clk @0ns=0 @12.345ns=1) )
after 5ns do(every 74.074ns do (wfm 13_5mhz @0ns=0 @37.037ns=1) )

h sr_ce pkt_head device reg_clr en pkt_cnt2
l sr_r reset write1 write2 write3 write
wfm pwrup @0ns=1 @35ns=0
wfm audio_clr @0ns=1 @80ns=0
wfm clr @0ns=1 @15ns=0

h pixeladdr[2:0]
l pixeladdr[23:3]

h a/v_data[23:20] a/v_data[15:12] a/v_data[7:4]
l a/v_data[19:16] a/v_data[11:8] a/v_data[3:0]


sim 1000ns
```

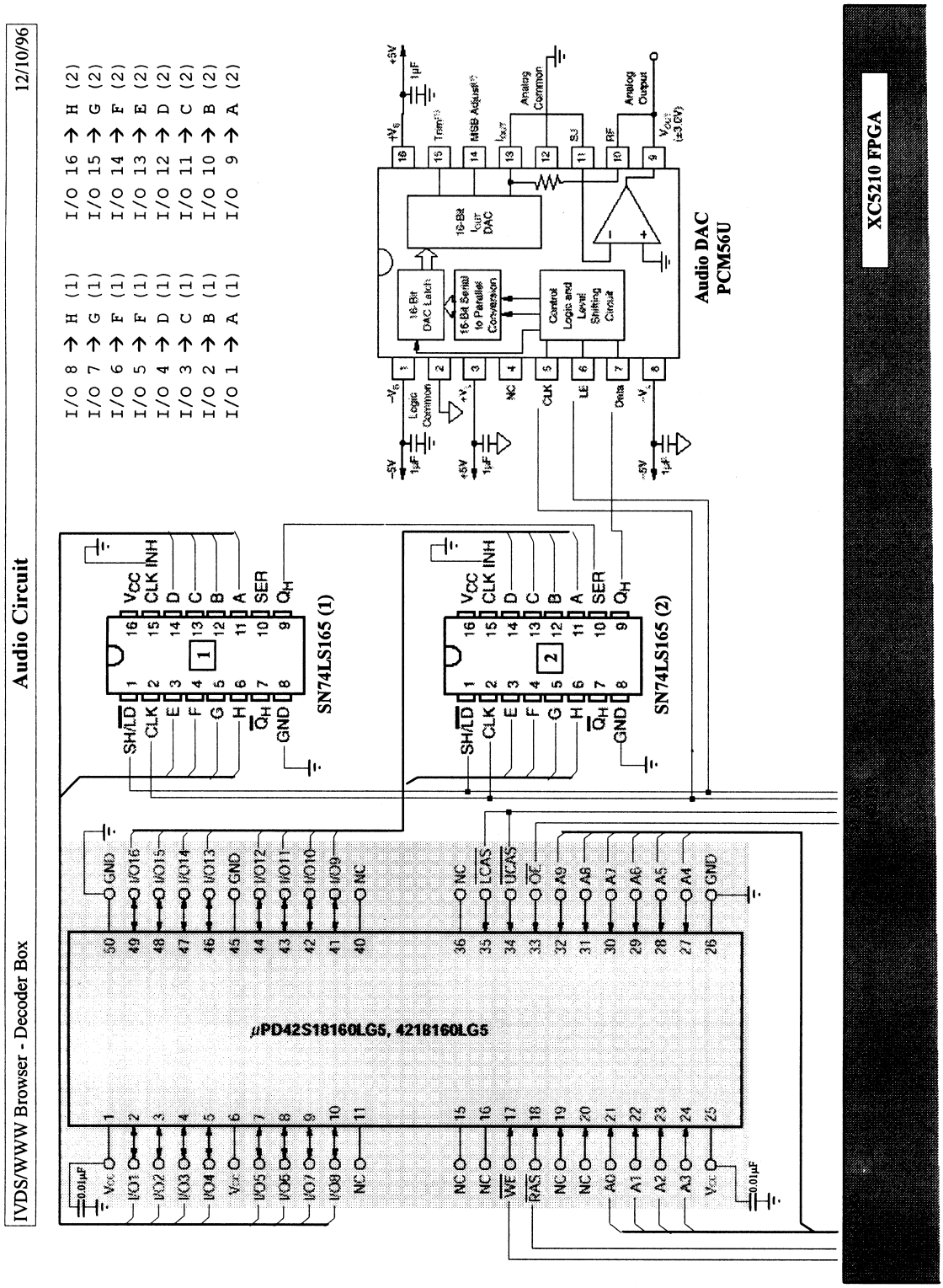

INT_CLK
13_5MHZ
UPDATE
A/V_DATA
OUT_UR
NTSC_DATA
PIXELADDR
NTSC_COUNT
NTSC_MATCH
F0F0F0
F0F0F0
F0F0F0
ZZZZZZ
ZZZZZZ
000007
00000
00001
00002
00003
00004
00005
00006
00007
00008
00009
0000A
0000B
0000C
0000D

Appendix D: Decoder Board Documents

D.1 Board-Level Schematic



D.2 Audio Circuit



D.3 FPGA Pin Connection Specification

XC5210-PQ208 FPGA
Pin Connections

Pin No	Pin Fct	Chip	P #	Pin Name
LEFT HAND SIDE				
1		NC		
2		GND		
3		NC		
4	PD7	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	19 10 6	I/O 7 I/O 8 H
5	PD6	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	18 9 5	I/O 6 I/O 7 G
6	PD5	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	17 8 4	I/O 5 I/O 6 F
7	PD4	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	16 7 3	I/O 4 I/O 5 E
8	PD3	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	15 5 14	I/O 3 I/O 4 D
9	PD2	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	13 4 13	I/O 2 I/O 3 C
10	PD1	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	12 3 12	I/O 1 I/O 2 B
11	PD0	HM62832U (**1**) µPD4218160G5 SN74LS165 (**1**)	11 2 11	I/O 0 I/O 1 A
12	PD15	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	19 49 6	I/O 7 I/O 16 H
13	PD14	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	18 48 5	I/O 6 I/O 15 G
14		GND		
15	PD13	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	17 47 4	I/O 5 I/O 14 F
16	PD12	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	16 46 3	I/O 4 I/O 13 E
17	PD11	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	15 44 14	I/O 3 I/O 12 D
18	PD10	HM62832U (**2**) µPD4218160G5 SN74LS165 (**2**)	14 43 13	I/O 2 I/O 11 C
19	PD9	HM62832U (**2**) µPD4218160G5	13 42	I/O 1 I/O 10

		SN74LS165 (**2**)	12	B
20	PD8	HM62832U (**2**) μPD4218160G5 SN74LS165 (**2**)	12 41 11	I/O 0 I/O 9 A
21	PD23	HM62832U (**3**)	19	I/O 7
22	PD22	HM62832U (**3**)	18	I/O 6
23	PD21	HM62832U (**3**)	17	I/O 5
24	PD20	HM62832U (**3**)	16	I/O 4
25		GND		
26		VCC		
27	PD19	HM62832U (**3**)	15	I/O 3
28	PD18	HM62832U (**3**)	14	I/O 2
29	PD17	HM62832U (**3**)	13	I/O 1
30	PD16	HM62832U (**3**)	12	I/O 0
31	PA7	HM62832U (**4**)	19	I/O 7
32	PA6	HM62832U (**4**)	18	I/O 6
33	PA5	HM62832U (**4**)	17	I/O 5
34	PA4	HM62832U (**4**)	16	I/O 4
35	PA3	HM62832U (**4**)	15	I/O 3
36		GND		
37	PA2	HM62832U (**4**)	13	I/O 2
38	PA1	HM62832U (**4**)	12	I/O 1
39	PA0	HM62832U (**4**)	11	I/O 0
40	PA15	HM62832U (**5**)	19	I/O 7
41	PA14	HM62832U (**5**)	18	I/O 6
42	PA13	HM62832U (**5**)	17	I/O 5
43	PA12	HM62832U (**5**)	16	I/O 4
44	PA11	HM62832U (**5**)	15	I/O 3
45	PA10	HM62832U (**5**)	14	I/O 2
46	PA9	HM62832U (**5**)	13	I/O 1
47	PA8	HM62832U (**5**)	12	I/O 0
48	PA23	HM62832U (**6**)	19	I/O 7
49		GND		
50	PA22	HM62832U (**6**)	18	I/O 6
51		NC		
52		NC		
BOTTOM SIDE				
53		NC		
54		NC		
55		VCC		
56	PA21	HM62832U (**6**)	17	I/O 5
57	PA20	HM62832U (**6**)	16	I/O 4
58	PA19	HM62832U (**6**)	15	I/O 3
59	PA18	HM62832U (**6**)	14	I/O 2
60	PA17	HM62832U (**6**)	13	I/O 1
61	PA16	HM62832U (**6**)	12	I/O 0
62	CS2A	HM62832U (**4,5,6**)	20	CS’
63	CS2B	HM62832U (**1,2,3**)	20	CS’
64	VA9	μPD424800 (**1,2,3**)	9	A9
65	VA8	μPD424800 (**1,2,3**)	10	A0
66	VA7	μPD424800 (**1,2,3**)	11	A1
67		GND		
68	VA6	μPD424800 (**1,2,3**)	12	A2
69	VA5	μPD424800 (**1,2,3**)	13	A3

70	VA4	µPD424800 (**1,2,3**)	16	A4
71	VA3	µPD424800 (**1,2,3**)	17	A5
72	VA2	µPD424800 (**1,2,3**)	18	A6
73	VA1	µPD424800 (**1,2,3**)	19	A7
74	VA0	µPD424800 (**1,2,3**)	20	A8
75	WE’	µPD424800 (**1,2,3**)	7	WE’
76	RAS’	µPD424800 (**1,2,3**)	8	RAS’
77	CAS’ INIT’	µPD424800 (**1,2,3**) XC17256D	22 3	CAS’ OE/RST’
78		GND		
79		VCC		
80	OE’	µPD424800 (**1,2,3**)	23	OE’
81	VD19	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	5 23 9	I/O 4 D19 Q3
82	VD18	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	4 24 8	I/O 3 D18 Q2
83	VD17	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	3 25 7	I/O 2 D17 Q1
84	VD16	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	2 26 6	I/O 1 D16 Q0
85	VD23	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	27 18 14	I/O 8 D23 Q7
86	VD22	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	26 19 13	I/O 7 D22 Q6
87	VD21	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	25 20 12	I/O 6 D21 Q5
88	VD20	µPD424800 (**3**) TMS320AV420 Am27S23(**3**)	24 21 11	I/O 5 D20 Q4
89	VD11	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	5 33 9	I/O 4 D11 Q3
90		GND		
91	VD10	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	4 34 8	I/O 3 D10 Q2
92	VD9	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	3 35 7	I/O 2 D9 Q1
93	VD8	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	2 36 6	I/O 1 D8 Q0
94	VD15	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	27 27 14	I/O 8 D15 Q7
95	VD14	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	26 29 13	I/O 7 D14 Q6
96	VD13	µPD424800 (**2**)	25	I/O 6

		TMS320AV420 Am27S23(**2**)	30 12	D13 Q5
97	VD12	µPD424800 (**2**) TMS320AV420 Am27S23(**2**)	24 31 11	I/O 5 D12 Q4
98	VD3	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	5 46 9	I/O 4 D3 Q3
99	VD2	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	4 47 8	I/O 3 D2 Q2
100	VD1	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	3 48 7	I/O 2 D1 Q1
101		GND		
102		NC		
103	DONE	XC17256D	4	CE’
104		NC		
RIGHT HAND SIDE				
105		NC		
106		VCC		
107		NC		
108	PROG’			
109	VD0	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	2 49 6	I/O 1 D0 Q0
110	VD7	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	27 39 14	I/O 8 D7 Q7
111	VD6	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	26 42 13	I/O 7 D6 Q6
112	VD5	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	25 43 12	I/O 6 D5 Q5
113	VD4	µPD424800 (**1**) TMS320AV420 Am27S23(**1**)	24 44 11	I/O 5 D4 Q4
114	V-INTER	TMS320AV420	51	INTER’
115	V- VSYNC	TMS320AV420	50	VSYNC
116	V-HSYNC	TMS320AV420	40	HSYNC’
117	V-CLK135	TMS320AV420	38	CLK135
118				
119		GND		
120				
121	CLK27	External Clock		
122				
123				
124				
125				
126	PP A4	Am27S23 (**1,2,3**)	5	A4
127	PP A3	Am27S23 (**1,2,3**)	4	A3
128	PP A2	Am27S23 (**1,2,3**)	3	A2
129	PP A1	Am27S23 (**1,2,3**)	2	A1
130		VCC		
131		GND		

132	PP A0	Am27S23 (**1,2,3**)	1	A0
133	PP A7	Am27S23 (**1,2,3**)	19	A7
134	PP A6	Am27S23 (**1,2,3**)	18	A6
135	PP A5	Am27S23 (**1,2,3**)	17	A5
136	PP G	Am27S23 (**1,2,3**)	16 15	G2 G1
137	CLK405	External Clock		
138	Au LE	SN74LS165 (**1,2**) AD1851	1 6	SH/LD' LE
139	Au CLK	SN74LS165 (**1,2**) AD1851	2 5	CLK CLK
140	Au CAS'	µPD4218160G5	34 35	UCAS' LCAS'
141	Au OE'	µPD4218160G5	33	OE'
142		GND		
143	Au A9	µPD4218160G5	32	A9
144	Au A8	µPD4218160G5	31	A8
145	Au A7	µPD4218160G5	30	A7
146	Au A6	µPD4218160G5	29	A6
147	Au A5	µPD4218160G5	28	A5
148	Au A4	µPD4218160G5	27	A4
149	Au A3	µPD4218160G5	24	A3
150	Au A2	µPD4218160G5	23	A2
151	Au A1 DIN	µPD4218160G5 XC17256D	22 1	A1 DATA
152	Au A0	µPD4218160G5	21	A0
153	CCLK	XC17256D	2	CLK
154		VCC		
155		NC		
156		NC		
TOP SIDE				
157		NC		
158		NC		
159	Au RAS'	µPD4218160G5	18	RAS'
160		GND		
161	Au WE'	µPD4218160G5	17	WE'
162	ID 0	ID DIP Switch		BIT 0
163	ID 1	ID DIP Switch		BIT 1
164	ID 2	ID DIP Switch		BIT 2
165	ID 3	ID DIP Switch		BIT 3
166	ID 4	ID DIP Switch		BIT 4
167	ID 5	ID DIP Switch		BIT 5
168	ID 6	ID DIP Switch		BIT 6
169	ID 7	ID DIP Switch		BIT 7
170				
171		GND		
172				
173	FE CRS	DP83840VCE	66	CRS
174	FE DV	DP83840VCE	64	RX_DV
175	FE ER	DP83840VCE	63	RX_ER
176	FE CLK	DP83840VCE	62	RX_CLK
177	FE D0	DP83840VCE	58	RXD0
178	FE D1	DP83840VCE	57	RXD1
179	FE D2	DP83840VCE	56	RXD2
180	FE D3	DP83840VCE	55	RXD3

181	MA0	HM62832U (**1 6)**	10	A0
182		GND		
183		VCC		
184	MA1	HM62832U (**1 6)**	9	A1
185	MA2	HM62832U (**1 6)**	8	A2
186	MA3	HM62832U (**1 6)**	7	A3
187	MA4	HM62832U (**1 6)**	6	A4
188	MA5	HM62832U (**1 6)**	5	A5
189	MA6	HM62832U (**1 6)**	4	A6
190	MA7	HM62832U (**1 6)**	3	A7
191	MA12	HM62832U (**1 6)**	2	A12
192	MA14	HM62832U (**1 6)**	1	A14
193	MA13	HM62832U (**1 6)**	26	A13
194		GND		
195	MA8	HM62832U (**1 6)**	25	A8
196	MA9	HM62832U (**1 6)**	24	A9
197	MA11	HM62832U (**1 6)**	23	A11
198	MA10	HM62832U (**1 6)**	21	A10
199	PA WE’	HM62832U (**4,5,6)**	27	WE’
200	PA OE’	HM62832U (**4,5,6)**	26	OE’
201	PA CS’	HM62832U (**4,5,6)**	25	CS’
202	PD WE’	HM62832U (**1,2,3)**	27	WE’
203	PD OE’	HM62832U (**1,2,3)**	26	OE’
204	PD CS’	HM62832U (**1,2,3)**	25	CS’
205		VCC		
206		NC		
207		NC		
208		NC		

FPGA Pin Identification

ID	Description
PD xx	Pixel Data pin xx
PA xx	Pixel Address pin xx
CS2B	Chip Select 2nd Set A
CS2A	Chip Select 2nd Set B
VA xx	Video Address pin xx
VD xx	Video Data pin xx
Au Ax	Audio Address pin x
Au Dxx	Audio Data pin xx
ID xx	ID Dip Switch pin xx
FE Dx	Fast Ethernet Data pin x
MA xx	Main memory Address pin xx
PP Ax	Palette PROM Address pin x
PP Dx	Palette PROM Data pin x

Notes:

1. There are 8 free I/O pins (2 at the Top side and 6 at the Left-hand side of the FPGA).
2. Pins with multiple connections imply that different chips use the same line (bus) and do not necessarily need to be connected to the FPGA pin.
3. Serial Configuration ROM (XC17256D) does not require (“consume”) I/O pins after configuration. Shaded entries in the above table indicate the connections required for the FPGA configuration.

4. The order of the FPGA pins (especially for the Address and Data pins) was selected having in mind the pinout of each external chip. Some advice from Kel Tech is expected, to eliminate any line crossings.

Vita

Aaron Hawes was born a United States Citizen on May 1, 1972 in Arlington, Virginia. He moved to Vienna, Virginia in 1979. He attended James Madison High School where he graduated in June, 1990. He graduated with a Bachelor of Science in Electrical Engineering at Virginia Tech in 1995. As an undergraduate, he co-oped for two eight month periods with Dynatech Communications based in Woodbridge, Virginia.

He graduated with a Master of Science in Electrical Engineering in 1997. As a graduate student, he held two research assistant positions. The first was with the Center for Wireless Telecommunications working on the IVDS World Wide Web Browser Project on which his thesis was based. The second was on a network modeling and simulation project funded by Newport News Shipbuilding. He focused on coursework in the Computer Area and had a strong interest in Computer Networking and the Internet.