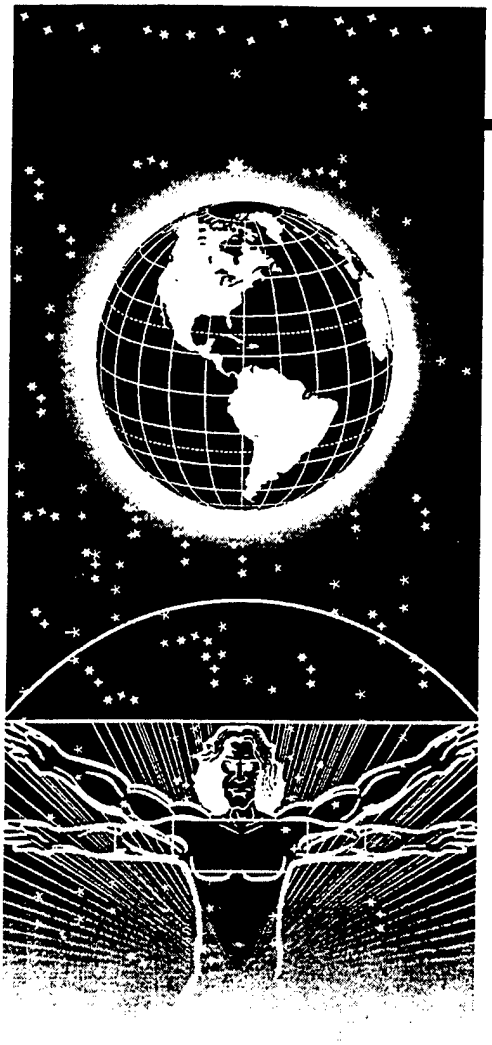
AFRL-HE-WP-TR-2001-0120



UNITED STATES AIR FORCE RESEARCH LABORATORY

DARPA

ADVANCED DISPLAY INTERFACE FOR OPEN SYSTEM HIGH RESOLUTION MILITARY AND COMMERCIAL APPLICATIONS

David J. Hermann
Daniel A. Perkins

BATTELLE MEMORIAL INSTITUTE
ELECTRONICS AND AVIONICS SYSTEMS
COLUMBUS OH 43201-2693

JUNE 2001

20010906 015

FINAL REPORT FOR THE PERIOD 23 SEPTEMBER 1998 TO 1 APRIL 2001

Approved for public release, distribution unlimited.

Human Effectiveness Directorate
Crew System Interface Division
2255 H Street
Wright-Patterson AFB, OH 45433-7022

NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Air Force Research Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Road, Suite 0944
Ft. Belvoir, Virginia 22060-6218

DISCLAIMER

This Technical Report is published as received and has not been edited by the Air Force Research Laboratory, Human Effectiveness Directorate.

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-2001-0120

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

MARIS M. VIKMANIS
Chief, Crew System Interface Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2001	FINAL REPORT 23 September 1998 - 1 April 2001

4. TITLE AND SUBTITLE

ADVANCED DISPLAY INTERFACE FOR OPEN SYSTEM HIGH RESOLUTION MILITARY AND COMMERCIAL APPLICATIONS

5. FUNDING NUMBERS

C: F33615-98-C-6003
PE: 62708E
PR: ARPH
TA: EC
WU: 01

6. AUTHOR(S)

David J. Hermann, Daniel A. Perkins

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

Battelle Memorial Institute
Electronics and Avionics Systems
Columbus OH 43201-2693

8. PERFORMING ORGANIZATION

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

Air Force Research Laboratory
Human Effectiveness Directorate
Crew System Interface Division
Air Force Materiel Command
Wright-Patterson AFB OH 45433-7022

10. SPONSORING/MONITORING

AFRL-HE-WP-TR-2001-0120

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Commercial display interfaces are transitioning from analog to digital format. Although this transition is in the early stages, the military needs to begin planning its own transition to digital. There are many problems with analog interfaces in high-resolution display systems that can be resolved by changing to a digital interface. Also, lower display system cost can be achieved by implementing a digital interface to a high-resolution display rather than an analog interface. The Advanced Display Interface (ADI) is designed to replace the analog RGB interfaces currently used in high definition workstation displays. The goal is to create a standard digital display interface for military applications that is based on current commercial standards. Support for military application-specific functionality is addressed, including display test and control. The main challenges to implementing a digital display interface are described, along with approaches to address the problems. Conceptual ADI architectures are described and contrasted. The current commercial standards for digital display interfaces are reviewed in detail. Finally, a demonstration system based on the chosen ADI architecture is described.

14. SUBJECT TERMS

Digital Video, Display Interface, Digital Display, Video Interface, Legacy Video, Digital Video Interface, ADI

15. NUMBER OF PAGES
109

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

CONTENTS

ABSTRACT (SF 298)....i

LIST OF FIGURES....iv

LIST OF TABLES....iv

FOREWORD....v

PREFACE....vi

ACKNOWLEDGEMENTS....vi

1. SUMMARY....1

2. INTRODUCTION....2

3. METHODOLOGY....6

4. RESULTS AND DISCUSSION....10

5. CONCLUSIONS....25

6. RECOMMENDATIONS....26

7. LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS....27

8. REFERENCES....30

APPENDICES
A. TRANSMITTER SCHEMATIC....33
B. RECEIVER SCHEMATIC....41
C. TRANSMITTER LAYOUT....49
D. RECEIVER LAYOUT....51
E. TRANSMITTER VERY HIGH LEVEL DESCRIPTION LANGUAGE....53
F. RECEIVER VERY HIGH LEVEL DESCRIPTION LANGUAGE....77
G. LIST OF PUBLICATIONS RELATED TO THIS DARPA-SPONSORED PROGRAM ...105

LIST OF FIGURE CAPTIONS

Figure 1. Migration to a digital display interface 2

Figure 2. Demonstration system architecture 7

Figure 3. Transition minimized differential signaling repeater 7

Figure 4. Extended interface 8

Figure 5. Transition minimized differential signaling link architecture 13

Figure 6. Video Electronics Standards Association Plug & Display connector 14

Figure 7. Digital Flat Panel connector 14

Figure 8. Digital Video Interface bandwidth scalability 15

Figure 9. Digital Video Interface connector 16

Figure 10. Gigabit video interface architecture 18

Figure 11. Demonstration system 20

Figure 12. Advanced Display Interfacer transmitter 23

Figure 13. Advanced Display Interface receiver 24

LIST OF TABLE TITLES

Table 1. Display categories 10

Table 2. Typical environmental requirements 11

Table 3. Transition minimized differential signaling bandwidth 13

Table 4. Personal computer configuration 21

Table 5. Flat panel display specifications 22

Table 6. Cable specifications 22

FOREWORD

This Contract F33615-98-C-6003 (ASTARS[1] WU[2] ARPHEC01) entitled "Advanced Display Interface for Open System High Resolution Military and Commercial Applications," was selected under Defense Advanced Research Projects Agency (DARPA) Broad Agency Announcement (BAA) 97-42, entitled "High Definition Systems (HDS)." This contract between the government and Battelle Memorial Institute (BMI) required the government to provide 100% of the total project funding of $102,703 under DARPA Order A940/47 to Dr. Hopper at the Air Force Research Laboratory (AFRL) as DARPA Agent. Mr. Reginald Daniels of AFRL served as the contract monitor. Performance was over the period 23 September 1998 through 1 April 2001. The program was slowed down by Battelle during the effort because of (a) dramatic activity in commercial digital interface standards during 1999-2000 and (b) the need for key Battelle personnel to provide engineering support to the Air Force Warner-Robins Air Logistics Center (WR-ALC) during their acquisition phase of the Common Large Area Display Set (CLADS) program in 1999-2000. Most of the present effort was accomplished during the period January-April 2001. Thus, the data are current even given the dynamic activity in digital standards in military and civil applications.

This report has been formatted in accordance with a commercial standard, with tailoring from the AFRL Scientific Technical Information Office. That standard is: "Scientific and Technical Reports—Elements, Organization, and Design," American National Standard ANSI/NISO Z39.18-1995 (NISO Press, Bethesda MD, 1995), available electronically via the following website address: http://www.wrs.afrl.af.mil/library/sti-pubh.htm

This report was edited by several AFRL personnel. This editing included a complete re-formatting of the draft provided by Battelle Memorial Institute on 3 May 2001 to comply with the aforementioned standard in the front matter, report body, and the first page for each appendix. In addition, the front matter sections were re-written and a three-page acronyms section was created. An Appendix G was added for publications produced with support from the present contract.

Appendices A-F have been retained in the reduced image format provided by Battelle. The drawings comprising these appendices are too large to present at larger resolution in the print version of this report. An electronic version this technical report will be made available to readers who may then view enlarged images of pages in these appendices via electronic magnification on their computers. The electronic version will be posted on an AFRL web site, www.hec.afrl.af.mil , with links on several other web sites, including Battelle Memorial Institute, www.battelle.org , and the Society for Information Display (SID), www.sid.org .

Technical review of this document was accomplished by Dr. Darrel G. Hopper, Mr. Reginald Daniels, and Mr. Frederick Meyer of AFRL and Mr. James Byrd of the Aeronautical Systems Center (ASC). Only minor changes, approved by Battelle, were made to technical content.

[1] A Science and Technology Activity Reporting System (ASTARS)
[2] Workunit (WU)

PREFACE

This program leverages the digital interface developed by Battelle under the WR-ALC Design Engineering Contract for the CLADS program for dim-ambient airborne mission crewstations in systems including the E-3 Airborne Warning and Control System, the E-8 Joint Surveillance Target Acquisition Reconnaissance System, C-130 tactical Airborne Command, Control and Communications system, and, possibly, the Airborne Laser System (see References 1 and 2). The program manager for CLADS is Mr. Bob Zwitch of WR-ALC. Several AFRL personnel (Dr. Darrel G. Hopper and Messrs. Frederick M. Meyer and Reginald Daniels) have consulted on CLADS since its inception.

The goal of this effort was to advance the development and adoption of the Advanced Display Interface (ADI) standard initiated under the CLADS program at WR-ALC.

There were several objectives of the ADI that are unique to military display applications. The interface must support high-resolution workstation graphics in industry-standard formats ranging from the video graphics array (VGA, 640 x 480 pixels), super VGA (SVGA, 800 x 600 pixels), extended graphics array (XGA, 1024 x 768 pixels), super XGA (1280 x 1024 pixels), wide-SXGA (W-SXGA, 1440 x 1024 pixel), to ultra-XGA (UXGA, 1600 x 1200 pixels).[3] In addition to these resolution requirements, the ADI must support both monochrome (8-bit) and color displays (18-bit to 24-bit) at 15 to 30 frames per second (fps). The intent of the ADI is to replace the interface between graphics card and display, so compatibility with emerging commercial standards for commercial off-the-shelf (COTS) graphics cards and graphic accelerator chips is required. The interface must provide this high data bandwidth without adding artifacts or degradation, potentially excluding the Motion Pictures Experts Group (MPEG) or other compression schemes. The digital interface should support extended cable lengths (up to 100 ft) for the many military applications where the graphics generators are located remotely from the displays. Also, the interface must support legacy video cabling, including miniature coax. The Digital Video Interface (DVI) standard, for example, limits cables to ten meters and does not support the cable types that exist in legacy systems. Finally, the interface should include all required functionality to specifically support military applications, including functionality not normally supported by a standard commercial interface, such as display built-in test (BIT) and display control, such as BIT initiate and reporting, display enable, brightness and contrast.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support received from DARPA under Contract Number F33615-98-C-6003. We thank Dr. Darrel G. Hopper, and Mr. Reginald Daniels of the AFRL for their support throughout the project.

---

[3] Editors note: Additional formats and far higher resolutions must now be added to the list for military application to leverage the rapidly advancing commercial state-of-the-art. First, several formats for digital high definition television (HDTV), ranging from 1280 x 768 to 1920 x 1080 pixels, must be considered given the deployment of HDTV broadcast just been initiated. Second, flat panel displays (FPDs) based on active matrix thin-film transistor (TFT) liquid crystal displays (LCD) became available in June 2001 with resolutions of quad-UXGA (QUXGA, 3,200 x 2,400 pixels) and QUXGA-wide (QUXGA-W, 3840 x 2400): the Toshiba 20.8 in. QUXGA TFT-LCD Monitor (com_net@jsh.thoshiba.co.jp) and the IBM 22.2-Inch QUXGA-Wide T220 Flat Panel Color Monitor (http://www2.ibmlink.ibm.com/cgi-bin/master?xh=zjCKPbPfZlFkpd1USenGnN9332&request=announcements&parms=H%5f101%2d178&xfr=N)

1. SUMMARY

As the world increasingly moves from analog to digital technology, the analog display interface is becoming obsolete. New flat panel display (FPD) technologies provide better performance at lower life cycle cost with a digital video interface. This is especially true for high-resolution displays, where the analog interface cost and undesirable artifacts both increase as resolution increases. When the analog display interface eventually becomes completely obsolete and unsupported in the commercial marketplace, the military will face a difficult technology obsolescence problem. Supporting the analog interface between ruggedized computers and FPDs well into the future will become expensive.

The military needs to make the transition to digital display interfaces. Displays implementing analog interfaces today should be incorporating open system architectures such as the Common Large Area Display Set (CLADS) to ease the transition to digital. More importantly, the military needs to adopt an open standard digital display interface. Without a standardized interface, each new digital FPD insertion will have a unique (and often proprietary) interface. Each new digital interface will be justified by optimizing it for a single new application. The many different 'standards' will continue to stovepipe military systems and increase maintainability costs while decreasing COTS leverage potential. By adopting a standard digital interface now, the military can break open existing closed systems and ensure that new systems cannot stovepipe all the way to the display. This will increase competitive opportunities in the military display business, resulting in lower display life cycle cost.

This report documents the initial effort to develop this standard, the Advanced Display Interface (ADI). Development is in the initial stages, beginning with an assessment of emerging commercial standards, enabling technologies, and military application requirements. The ADI standard is based on current commercial standards for digital display interfaces. The goal is to maximize COTS leverage potential when implementing ADI and display systems. In unison with this goal, every effort has been made to minimize COTS obsolescence by using a commercially successful standard (for example, the 15-pin VGA interface has been around since 1987). A demonstration system has been designed, and initial results have shown that it will be cost effective.

Further efforts are needed to complete the ADI standard. Support is needed from military users to develop, promote, and adopt the ADI standard.

2. INTRODUCTION

2.1 Background

Most high-resolution displays in use today are part of a computer workstation. These workstation displays are typically large, bulky, high-resolution cathode-ray tube (CRT) displays. Since the CRT is an analog device, these displays utilize an analog red, green, blue (RGB) video interface. The CRT-based displays also require analog display adjustments for a number of CRT-specific corrections (image sizing, centering, pincushion, trapezoid, rotation, keystone, etc.). The graphics generators in the current workstations are almost entirely digital circuits. The graphics card includes a random access memory digital-to-analog converter (RAMDAC) as the last stage of video generation to create an analog RGB output to interface to an analog CRT. The only purpose of the RAMDAC and analog interface is to provide the analog video signal required by the CRT.

The current trend is towards workstations with FPDs, as shown in Figure 1. These FPDs are characterized by completely digital electronics with direct-mapped pixels. Although the FPDs are high resolution, each pixel is individually addressed with digital accuracy. In the future workstations, the graphics cards will be completely digital, with no RAMDAC or analog components. A digital interface is used to provide digital video to the digital FPD for direct display without distortion or artifacts.

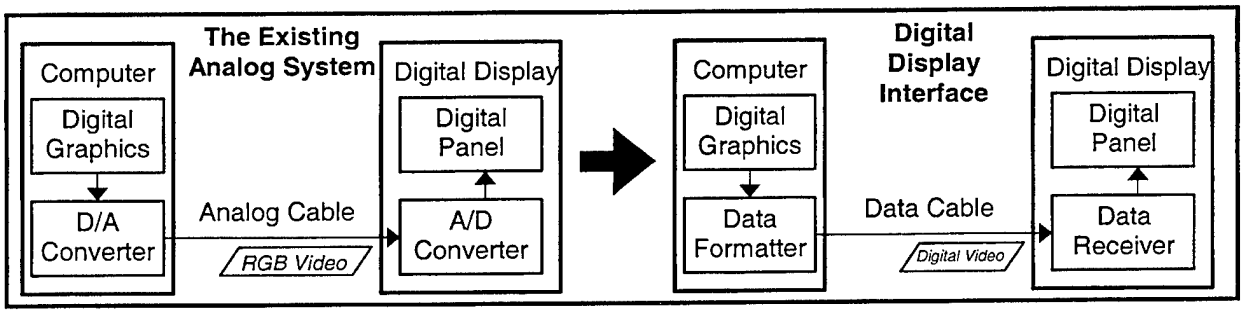


Figure 1. Migration to a digital display interface.

2.2 Advanced Display Interface

The ADI is an open standard digital display interface capable of supporting high-resolution displays in military applications. The ADI program builds on previous Battelle efforts with digital display interfaces. In 1996, Battelle developed one of the first practical high-resolution digital display interfaces for the Common Large Area Display Set (CLADS) program. The CLADS head assembly test set provides digitally-accurate test images and video at 1280 by 1024 resolution with 24-bit color at a fast 60 Hz refresh rate (Reference 1). This test set consists of a digital graphics card and digital Application Video Interface Module (AVIM). The digital graphics card is a standard personal computer interface (PCI) graphics card that runs under Microsoft Windows® with standard graphics card software drivers. The digital AVIM receives the graphic card digital video and reformats it for the standard DVI used by the *display* head.

The graphics card and digital AVIM are connected by multiple copper Fibre Channel[4] links, allowing up to 75 feet between the test computer and display under test (Reference 2). Also for the CLADS program, Battelle developed an open standard High Speed Digital Display Interface (HSDDI). This interface is used between a modular AVIM and the head assembly within a CLADS unit. The HSDDI provides 24-bit digital video at 1280 by 1024 resolution and beyond (Reference 3).

The ADI concept is consistent with recommendations of the former DoD Large Display Work Group (LDWG) Interfaces Subgroup. DoD advocates the use of an open standard digital interface between graphics generator and display device for both intelligent and 'dumb' displays (Reference 4).

The ADI is based on emerging commercial digital display interface open standards. The most prominent commercial digial video interfaces are based on transition minimized differential signaling (TMDS). These standards include the Video Electronics Standards Association (VESA) Plug & Display (P&D) standard (Reference 5), the VESA Digital Flat Panel (DFP) standard (Reference 6), and the Digital Display Working Group (DDWG) Digital Visual Interface (DVI) (Reference 7). All of these commercial standards are very similar and all use the PanelLink™ TMDS technology. In addition to these medium cable length TMDS interfaces, ADI will consider much longer cable applications using the Fibre Channel standard physical layer interface (Reference 8).

The ADI program can provide immediate benefits to military programs with difficult display interface problems. The ADI may be a potential solution for the E-3 Airborne Warning and Control System (AWACS) upgrade problems. In this application, the displays (currently being upgraded to digital technology) are up to 100 feet from the graphics generators (also being upgraded). These must be connected using existing cabling, which consists of four mini-coax cables (type RG-179B) and a twin axial (twinax) sync cable. The ADI can also provide a solution for a number of other avionics displays that Battelle is currently evaluating, especially in cases where the displays are not located near the host computers.

2.3 Video Interface Problem

Many of the current problems with today's display systems originate from the analog RGB interface. This interface was developed primarily to interface to analog CRT technology, for which it is well suited; however, the analog RGB must be digitized for use with FPDs. This creates several difficult problems that are not cheaply solved. First, the analog video must be sampled using high-speed analog-to-digital converters (ADCs). With high-resolution displays, the sample clock can easily be over 120 MHz, requiring three costly ADCs (for full-color) to recover the original digital signal. Due to bandwidth considerations and signal-to-noise limitations of the analog system and high-speed ADCs, only about six of the original eight bits of gray scale can be accurately recovered.

Another difficult problem involves accurate time sampling of the original graphics pixels. The ADC does not know the exact locations of the pixels because it does not have the original pixel

[4] Fibre Channel is the name of a digital interface standard developed by the American National Standards Institute It is defined at http://www.ancor.com/fcinfo.htm and http://www.fibrechannel.com . Also, see Section 4.1.2.5 below.

clock available to it (this clock does not leave the graphics card in an analog RGB interface). An expensive and sensitive ultra-precision phase lock loop (PLL) is used to recover the pixel clock from the video sync information. The PLL makes a best guess at the pixel locations, but there is always an error associated with the PLL sample clock. This error, or clock jitter, causes aliasing errors in the resulting digitized video. These aliasing errors cause visual artifacts on the FPD, reducing the display system performance.

Another problem with the analog interface is accurately centering the digitized video on the FPD. Since the FPD is a digital device, it typically has exactly the number of pixels available on the display that are required for the given graphics resolution. If the image is not exactly centered both horizontally and vertically, image pixels will be lost.

The solution to all these problems is to use a digital display interface instead of analog RGB. Only a digital interface will provide full performance for digital FPDs.

2.4 Digital Interface Objectives

There are several objectives for a military digital display interface to meet. Obviously, the interface must replace analog RGB video with digital video, without adding artifacts or degradation. The digital video interface should be standardized and open, and should be based on or related to commercial standards such as DDWG DVI. The digital interface should support extended cable lengths for the many military applications where the graphics generators are located remotely from the displays. The interface should be rugged enough to meet military environmental requirements. Finally, the interface should include all required functionality to specifically support military applications, including functionality not normally supported by a standard commercial interface. This includes support for display BIT and display control, such as BIT initiate, BIT reporting, display enable, brightness control, and contrast control.

2.5 Challenges

There are several challenges to implementing the digital interface, centering on each of the three parts of the video interface system. Beginning at the video source, the graphics generator or graphics card raises certain legacy issues. Most graphics cards available today provide analog RGB output and do not include any digital video output.[5] The commercial graphics accelerator chips that these cards are based on contain internal RAMDACs, so the digital video raster does not exist outside the graphics chip.

The interface cable between the graphics generator and the display has potential problems. The TMDS physical interface does not support extended cable lengths. The DVI standard, for example, limits cables to ten meters. Other optional interfaces, such as the Universal Serial Bus (USB) and the Institute of Electrical and Electronics Engineers (IEEE) standard IEEE1394 (nickname: Firewire), further limit the VESA P&D standard to five meter cables (Reference 5). Also, TMDS is not supported for arbitrary cable types that may exist in legacy systems.

[5] It is still true in June 2001 that most graphics cards provide analog but not digital output, although there are many more graphics cards with DVI outputs than at the start of this project in 1998. Looking at the entire graphics card market, only the high end cards have DVI output, but if one wanted to put together a system with DVI output, one could obtain the necessary card from several vendors (at a premium price). Most presentation projectors and LCD displays have DVI input, but the statement here is related to DVI output on graphics cards.

Although there is an approach to providing TMDS signals over fiber optic cable, other cable types have not been addressed. There may also be environmental issues related to COTS TMDS support.

Finally, the FPD raises several interface issues. There is no standard BIT interface for military displays. Also, there is no standard display control interface. Although VESA Display Data Channel 2 (DDC2, Reference 9) provides a standard data interface to the display, this is primarily used for system initialization (data from display to graphics card only). The DDC2 was originally developed to support plug-and-play of displays, but it could be extended to include display controls such as brightness and contrast.

The solution to the challenges given above is to develop an Advanced Display Interface that addresses all these issues. The ADI will specifically support military and industrial users and applications. The ADI will be based on open commercial standards, as an extension of existing or emerging digital interface desktop standards. The ADI will, however, specifically address the unique needs of military users.

3. METHODOLOGY

The general approach of the ADI program is to leverage existing standards to improve military display systems by improving performance while reducing cost. This is accomplished through COTS technology insertion and maintaining an open systems environment. A primary objective is to avoid "reinventing the wheel," attained through utilization of commercial standards and COTS technology by extending the existing capabilities.

3.1 Technology Assessment

The first step towards developing an advanced display interface centers about defining the requirements and applications of such an interface. This is accomplished by an assessment of the current technology and future applications of displays. The technology assessment includes investigations of both commercial and military applications and the enabling technologies available for a digital video interface.

3.1.1 Display Requirements

Current and future display requirements for both military and commercial markets are reviewed for potential applications of ADI. This includes a survey of graphics modes for ADI, including resolution, colors, and refresh rates. An analysis of cost tolerance of ADI is considered for both military and commercial markets. Data sources for military applications include conference papers, military program offices, display vendors, and the Military Display Market comprehensive report published by AFRL (Reference 10).

3.1.2 Digital Interface Standards

Current and emerging display standards were researched for applicability to ADI. These include the VESA and Fibre Channel standards. Specific signaling and synchronization details of these standards are summarized into a set of guidelines for the development of the ADI standard. Also addressed are cable interconnect issues. For long cable runs, the ADI program explores methods to extend TMDS data to 100 feet. This also includes adapting to legacy system cable media, such as miniature coax or triax. For example, TMDS data could be converted into a Fibre Channel data stream and transmitted on RG-179B coax media. Such a system could provide digital video interface solution for E-3 AWACS, where four RG-179B cables connect each monitor to a graphics generator up to 100 feet away.

3.1.3 Enabling Technologies

Current and emerging technologies were researched for potential components in the implementation of ADI. Included are integrated circuits (ICs), graphics cards, cables, and software that may be used to implement ADI. Problems were addressed in each of the three parts of the video interface system: graphics card, cabling, and display. For graphics card issues, we examined emerging COTS graphics cards with digital interfaces. Most of the cabling issues are already addressed with the Digital Interface Standards. Display issues include integrated digital data receivers in the emerging flat panel displays.

3.2 Demonstration System

To demonstrate a digital video interface over long cables, Battelle designed and built a laboratory demonstration system. This included interfacing to the cable media type at both ends of the cable. The demonstration system has a digital video source, cable media driver, cable, cable media receiver, and display device, as shown in Figure 2. These demonstration system components were used to evaluate cable types and lengths and ADI data rates, resolutions, and color depths.

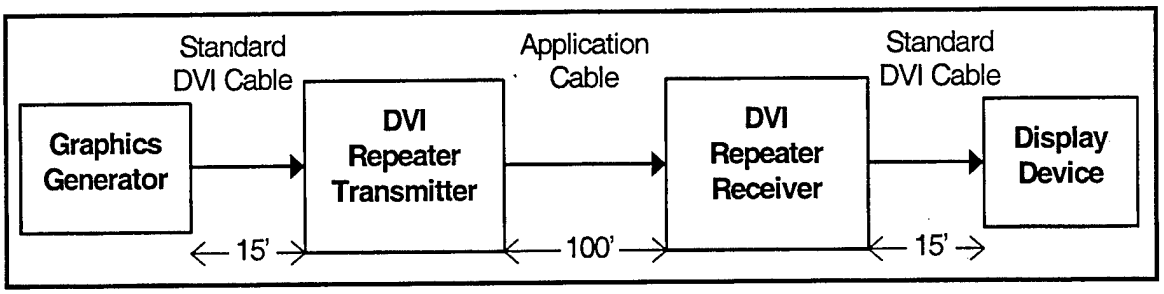


Figure 2. Demonstration system architecture.

There are a number of different possible architectures for an ADI system. To maximize the applications suitable for ADI insertion, the ADI is inherently flexible in physical layout and electrical hierarchy. Two sample architectures are shown in Figure 3 and Figure 4. The first is a simple TMDS repeater. The TMDS repeater is designed to provide a standard TMDS interface to both the graphics generator and the FPD. A transmitter and receiver pair converts the TMDS signals to a format compatible with the application cable. The transceiver hardware also provides the DDC link throughput. The transmitter and receiver could leverage other commercial data standards, such as Fibre Channel (Reference 8), to provide the data pipe on the legacy application cable over long distances. This approach provides maximum leverage of COTS components: the graphics card and FPD appear to be connected to each other via a standard TMDS cable. The repeater hardware provides an electrically transparent extension of the commercial standard TMDS interface.

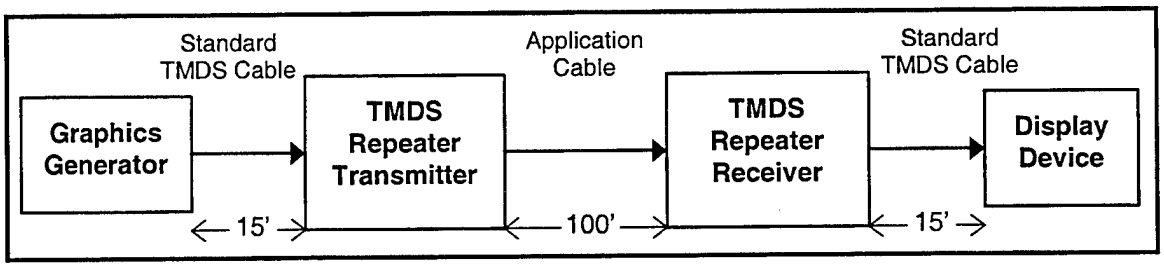


Figure 3. Transition minimized differential signaling repeater.

A second, more integrated, architecture is shown in Figure 4. This extended interface integrates the function of the repeater transmitter into the workstation computer and the function of the repeater receiver into the display. A standard mezzanine connector is defined for the graphics card, and a daughter card is inserted for the specific cable type used. The application cable then plugs directly into this daughter card. At the display end, an AVIM in a modular display (such as CLADS) interfaces directly with the application cable (Reference 2). The AVIM then converts the data into a standard format for use by the display device via a plug-in connector between the AVIM and display device. Although this provides a more integrated and clean solution, the components are electrically equivalent to the repeater design.

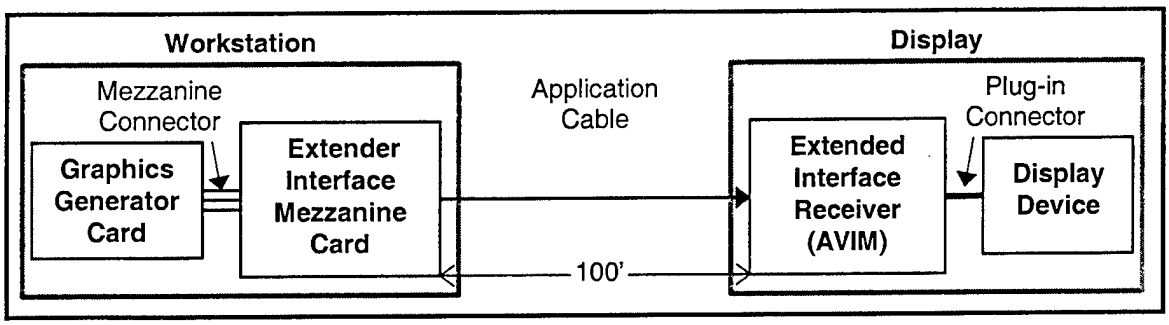


Figure 4. Extended interface.

3.2.1 Display Interface Standard

The display interface standard chosen for the demonstration system is the DVI. This was chosen based on the assessment of Digital Interface Standards and the commercial success of DVI. The DVI interface applies to both the graphics card and the display device. The DVI interface is based on TMDS.

3.2.1.1 Digital Video Source

The digital video source is a COTS Microsoft Windows® personal computer (PC) with a DVI output graphics card. This provides a low-cost and versatile digital graphics generator.

3.2.1.2 Flat Panel Display

The display device is a COTS active matrix liquid crystal display (AMLCD) with DVI input capability. This provides a low-cost high-quality output device to evaluate the performance of ADI.

3.2.2 Cable Interface

The cable interface is a standard Fibre Channel physical layer media interface. The specific supported media type is miniature coax, although twisted-pair, twinax, and optical fibre can also be supported with minor modifications of the media interface. The Fibre Channel output drivers are implemented using Cypress HOTLink transmitter and receiver integrated circuits. The Fibre Channel links are unidirectional, and none of the upper-level protocols are implemented. The Fibre Channel components are used to implement a low-cost reliable bit pipe without any of the

overhead required by a Fibre Channel network data link. The cable interface receives DVI data, converts it to Fibre Channel data, transports the data across coax, and converts it back to DVI.

3.2.2.1 Cable Media Driver

The cable media driver receives full-rate high-resolution digital video from a DVI graphics card. The TMDS data is received via a standard TMDS cable and connector. The cable driver performs data rate reduction by slowing down the data to 15 fps from 60 fps. The bit-depth is also reduced from 24-bit to 16-bit. The reduced bandwidth data is transmitted over one to four miniature coax cables to the receiver.

3.2.2.2 Legacy Cable

The legacy cable chosen for the demonstration system is RG-179B miniature 75 Ω coax. This cable type was chosen for maximum applicability to military aircraft display cabling. One to four cables can be utilized, with increased bandwidth available with more cables. The assessment of Display Requirements indicates that one to four cables may be present in a military display system.

3.2.2.3 Cable Media Receiver

The cable media receiver converts the reduced bandwidth Fibre Channel data from the transmitter to standard DVI data for the display. The receiver provides 24-bit, 60 fps DVI output to the display by zero-stuffing the truncated bits and sending each frame four times.

4. RESULTS AND DISCUSSION

A list of journal and proceedings publications reporting work accomplished under this DARPA-sponsored Cooperative Agreement in additional detail is provided in Appendix G.

4.1 Technical Assessment

The results of the initial technical assessment were originally published in Reference 11. That assessment material has been updated in this report (below) and the military display requirements have been added.

4.1.1 Display Requirements

Current military display systems can be grouped into three main size categories, based on required resolution. These groups are shown in Table 1.

Table 1. Display categories.

Category	Size	Resolution
Small	Up to 4 x 6 in.	Up to 640 x 480 (VGA)
Medium	4 x 6 in. to 6 x 8 in.	640 x 480 (VGA) to 1024 x 768 (XGA)
Large	Greater than 6 x 8 in.	1024 x 768 (XGA) to 1600 x 1200 (UXGA)

In addition to the resolution requirements shown in Table 1, military displays typically require either 256 gray shades (8-bits per monochrome pixel) for monochrome displays or between 64 and 256 gray shades (18-bits to 24-bits per color pixel) for color displays (Reference 12). Required frame rates vary from 30 frames per second (fps) for sensor video displays to 15 fps for workstation displays.

The typical environmental requirements of military display systems are given in Table 2 The components of the display interface must also meet these requirements. Environmental qualification should be in accordance with Radio Technical Commission for Aeronautics (RTCA) Document (DO) number DO-160C (Reference 13).

Table 2. Typical environmental requirements.

Parameter	Typical Requirement
Temperature	Operating: -54 to +55°C Short-term high: +70 °C Non-operating: -54 to +95 °C
Altitude	0 to 50,000 ft
Humidity	100% with condensation
Vibration	1.4 g RMS total
Shock	Operating: 20 g each axis (11ms) Crash: 40 g (impulse)
Explosive Atmosphere	Explosion proof
Sand, Dust, Salt Spray	Sealed
Fungus	Resistant
EMI *	MIL-STD-462D Class A1b (Reference 32)

* Electromagnetic interference

An auxiliary data channel is typically needed between the computer and the display. This data channel is used for BIT and control; it is bi-directional and must be reliable over the same long cables as the digital video. Standard open system protocols for both BIT and display controls will ease integration of FPDs into military systems. The protocols can be integrated with standard commercial monitor control interfaces such as VESA DDC2, Reference 9. The resulting merged control data (DDC2 with extensions) is transferred through ADI on a separate conductor or by piggybacking on the serial video data stream.

4.1.2 Digital Interface Standards

Currently, there is much activity in the commercial sector related to digital FPD interfaces. In the past few years, several different standards and methods for digitally interfacing FPDs to computers have been developed. From these ongoing efforts, several commercial standards are available for use as a basis for an ADI system; however, it is likely that only one standard will dominate the commercial marketplace and the others will be eventually abandoned. To provide the most COTS leverage, ADI needs to take advantage of the success of the winning commercial standard interface.

The first formally standardized digital FPD interface is the VESA P&D interface, Reference 5. This is documented in a detailed 109 page open standard first released June 11, 1997. This standard uses the PanelLink™ TMDS for the video data. A lower cost alternative to VESA P&D was then developed independently and later adopted by VESA, the Digital Flat Panel interface standard, Reference 6. The final standard was released by VESA on February 14, 1999, and is only 16 pages. It references VESA P&D for much of the detail, including the same TMDS interface. On April 2, 1999, the DDWG published the first release of the DVI standard, Reference 7. This detailed 76-page standard also uses the TMDS interface, and is backward compatible with both VESA P&D and DFP.

Another commercial standard data interface, Low Voltage Differential Signaling (LVDS), is available for use in digital video interfaces; however, there is no detailed standard for implementing LVDS FPD interfaces. Several solutions exist from LVDS silicon vendors in the form of applications notes, but no independent standards organization has adopted an LVDS FPD interface standard. The main LVDS standard includes the physical layer interface only, and is generic for any type of digital data pipe, not specifically for FPD interfacing.

There is also a Special Working Interest Group (SWIG) within the Fibre Channel Association that is working on an Audio/Video transport mapping for Fibre Channel (FC-AV). The current draft (Reference 15) is from August 1997, and contains high-level protocol and data mappings for television and MPEG. Another Fibre Channel SWIG is working on an Avionics Environment (FC-AE) for Fibre Channel. The current draft (Reference 16) is from April 1996, and contains Fibre Channel extensions to support avionics command, control, instrumentation, simulation, signal processing, and sensor/video data distribution. Boeing is pushing the Fibre Channel approach for both civil and military aircraft avionics (Reference 31).

Sony has developed an alternative FPD interface, the Gigabit Video Interface (GVIF), which remains proprietary. The GVIF uses considerably fewer conductors than TMDS, allowing a much thinner cable and miniature connectors to be used for portable applications (Reference 17).

4.1.2.1 Video Electronics Standards Association Plug & Display

The VESA P&D standard was developed by VESA members to provide an open standard for digitally interfacing FPDs to PCs. The VESA members are comprised of graphics card manufacturers and display manufacturers (325 member companies) with the following mission: "To promote and develop timely, relevant, open display and display interface standards, ensuring interoperability, and encouraging innovation and market growth." (Reference 18). The standard was released in June 1997 with support from the following companies and institutions: 3M, AMP, Canon, Chips and Technologies, Hewlett Packard, Hirose Electric, Hitachi, Hosiden, IBM, JAE, Madison Cable, Mitsubishi, Molex, National Semiconductor, NEC Technologies, the US Department of Commerce National Institute for Standards and Technology (NIST), Panasonic, Philips, Silicon Image, Sun Microsystems, Texas Instruments (TI), Toshiba, and others.

The P&D interface is designed to be a complete interface between PC and display, including legacy support for analog CRTs and optional data interfaces for future expansion. The P&D standard allows cable lengths up to 10 meters using TMDS signaling and a single connector for all supported interfaces. The included analog option consists of RGB video and pixel clock (to support analog interface FPDs).

The TMDS interface used in P&D is based on the VESA Flat Panel Display Interface (FPDI-2) developed for laptop computers. This interface is used in many laptops with resolutions of 1024 by 768 pixels or higher. The TMDS-based FPDI-2 interface uses the PanelLink™ technology to reduce EMI emissions and allows high-resolution laptops to pass testing required by the Federal Commuications Commission (FCC). Figure 5 shows the TMDS link architecture, and Table 3 shows some of the TMDS bandwidth data for high-resolution applications.

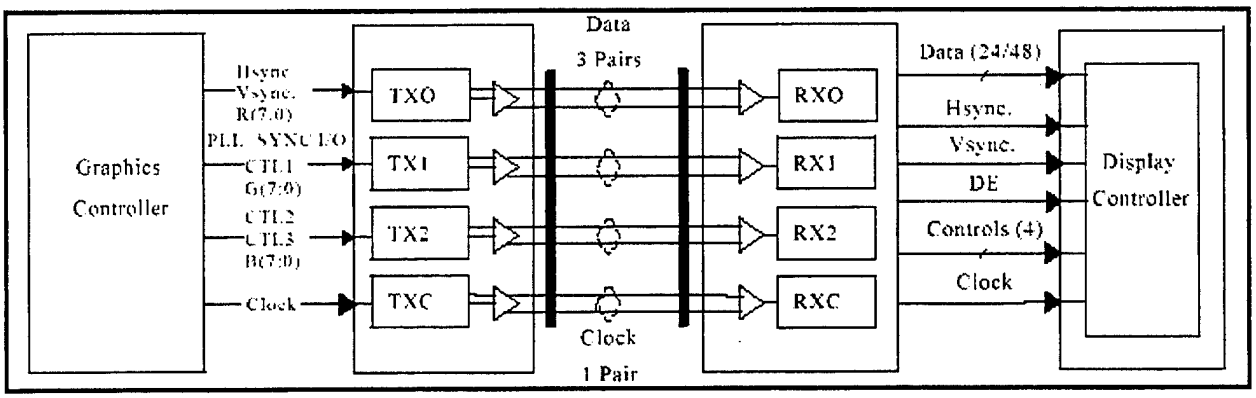


Figure 5. Transition minimized differential signaling link architecture (Reference 5).

Table 3. Transition minimized differential signaling bandwidth.

Resolution Horiz.	Vertical	Frame Rate (Hz)	Pixel Clock (MHz)	24-bit Data Rate (MB/s)	CRT Refresh (Hz)
1024	768	30	24	71	60
1280	1024	30	40	120	60
1600	1200	30	59	176	60
1920	1080	30	62	187	60 (interlaced)

* MB: megabyte

The P&D standard uses the VESA Display Data Channel (DDC2) for display initialization and control (Reference 9). This includes initialization logic in the display, support for the VESA Extended Display Identification Data (EDID) standard, support for the VESA Display Power Management Standard (DPMS), and support logic in the graphics adapter.

The P&D standard includes USB and IEEE 1394 (Firewire) as optional data interfaces that are carried along the same P&D interface cable. Use of either of these optional interfaces limits cables to five meters or less. With these optional interfaces, the display becomes a connectivity hub for the workstation computer. With USB, for example, the display becomes a USB hub with the mouse and keyboard plugging directly into connectors on the display. Other peripherals, such as printers, digital cameras, and scanners, can also be plugged directly into the display. This allows the workstation computer to be located on the floor or away from the display, without the need to wire everything into the back of the computer box. A single cable connects the display to the computer and provides the peripheral bus interconnection.

The P&D standard uses a single connector for all interface signals. The connector is shown in Figure 6. The P&D connector includes 30 data contact pins and four analog 75Ω pins (MicroCross™ quasi-coax).

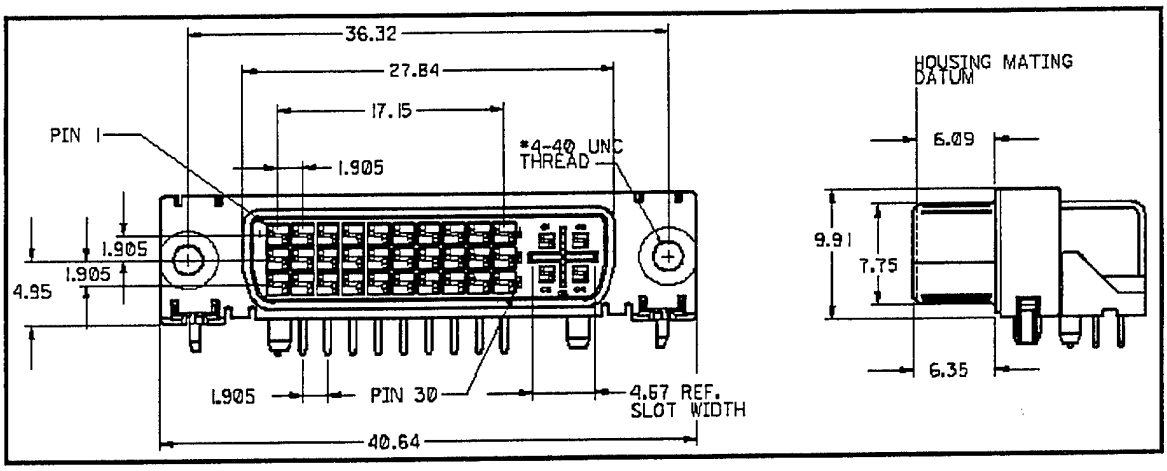


Figure 6. Video Electronics Standards Association Plug & Display connector (Reference 5).

4.1.2.2 Video Electronics Standard Association Digital Flat Panel

In 1998, an effort was made to develop a low-cost version of P&D to enable more rapid penetration of digital FPD interfaces into the PC marketplace. In May 1998, ATI Technologies, Inc.(ATI), a manufacturer of graphics cards, began a partnership to develop and promote the DFP port. The DFP Working Group, headed by Compaq, was formed to propose DFP as a VESA standard. Before the standard was adopted, Compaq began manufacturing and selling PCs and FPDs with the DFP interface (Reference 19). VESA released the final version of the standard in February 1999. The standard is endorsed by several companies, including 3dfx Interactive Inc., 3Dlabs, 3M, AMP, ATI, Compaq, Creative Labs, Matrox Graphics, Molex, Pixelworks, Princeton, S3, Taiko Denki, VideoLogic, and ViewSonic (Reference 20).

The DFP interface is a subset of VESA P&D with several features removed and a different connector. The standard, Reference 6, is very brief, and references the P&D standard for much of the details. It uses PanelLink™ TMDS for video data and VESA DDC2B for control. It does not include the analog option or optional data interfaces (USB and Firewire) of P&D. The control protocol includes VESA DDC2B support only, with EDID and DPMS functions. To simplify the data interface, only a single simple data mapping is used (P&D includes other data mappings to support Dual Scan Twisted Nematic (DSTN) LCD panels). The DFP interface uses a single connector with cables limited to five meters. The connector, shown in Figure 7, is a low-cost 20-pin mini-D ribbon (MDR) connector.

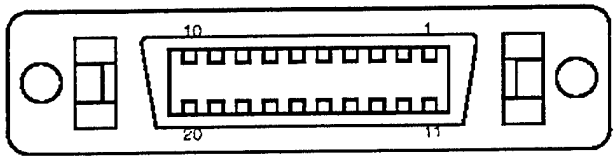


Figure 7. Digital Flat Panel connector (Reference 6).

4.1.2.3 Digital Displays Working Group Digital Visual Interface

In September 1998, Intel formed the DDWG to produce a single industry specification for the digital display interface (Reference 21). The goal of the DDWG is to eliminate the confusion caused by the many specifications and consortiums that currently exist for digital displays. The DDWG Promoters include Intel, Silicon Image, Compaq, Fujitsu, Hewlett-Packard, IBM, and NEC. Other prominent members include Microsoft, Dell, and Molex. The first release of the DVI, Reference 7, was in April 1999. DVI is based on 'Open Intellectual Property (IP),' a mutual agreement amongst companies to license patents and other IP necessary to implement the interface on a reciprocal, royalty free basis (Reference 22). This may free DVI implementers from any IP issues related to the patented PanelLink™ TMDS technology from Silicon Image. Although the DVI is independent from VESA, its members are VESA members and the DDWG intends to keep VESA informed of its activities and progress. The DDWG has made every effort to make DVI interoperable with both VESA P&D and DFP (using adapter cables).

DVI is based on both P&D and DFP, and uses the same PanelLink™ TMDS interface. It also uses VESA DDC2B for control and references several other VESA specifications for video timing and signals on the analog option of DVI. DVI, however, differs from P&D in that it allows for additional TMDS links to increase interface bandwidth. With dual TMDS links (six data pairs and a clock pair), DVI supports resolutions beyond 2K by 2K (max. 350 MHz pixel clock). DVI bandwidth scalability is shown in Figure 8.

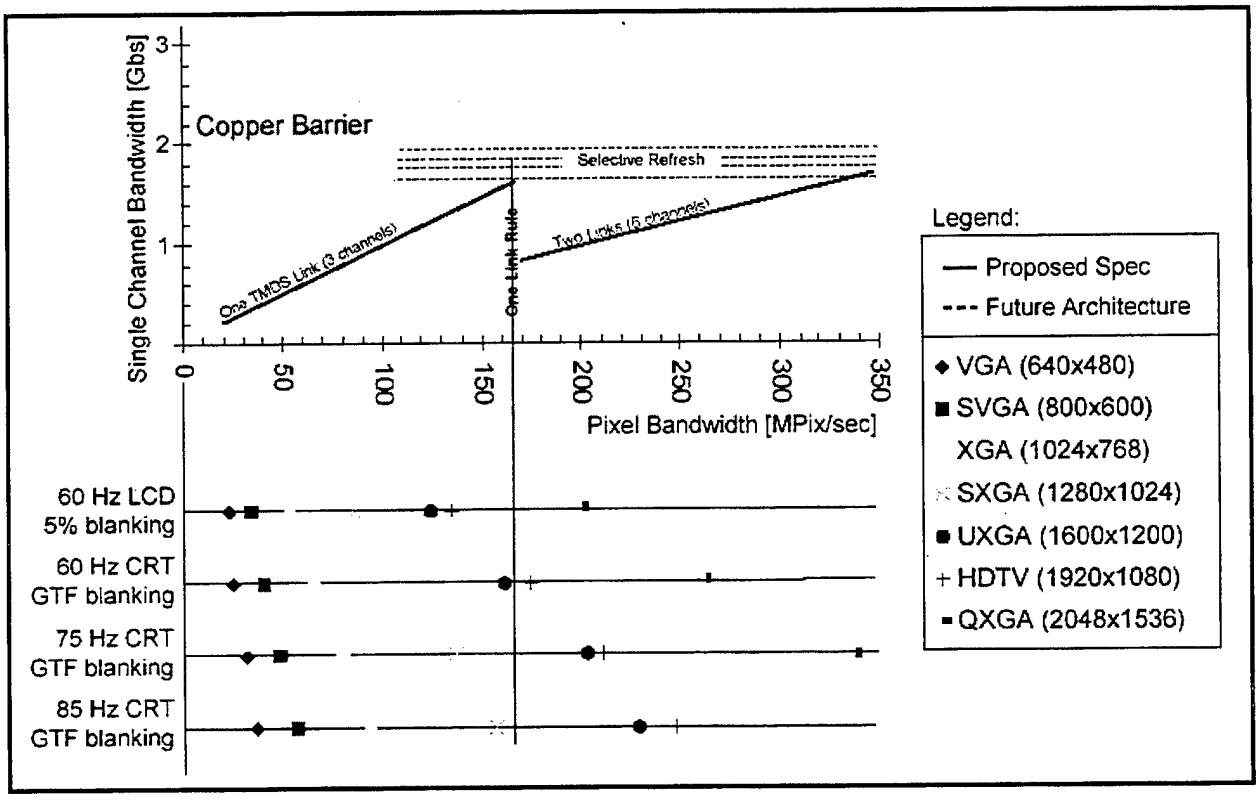


Figure 8. Digital Video Interface bandwidth scalability (Reference 7).

Cable lengths are similar to other TMDS interfaces, about ten meters maximum. DVI does not support any optional interfaces such as USB or Firewire. DVI uses a single connector, shown in Figure 9. The connector includes 24 data pins and four quasi-coax connections. A digital-only version omits the four quasi-coax signals.

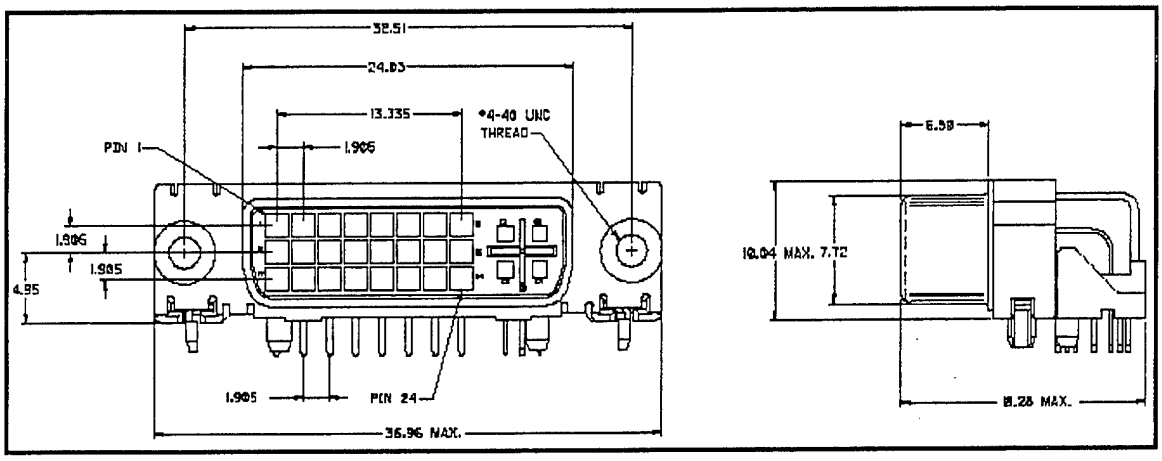


Figure 9. Digital Video Interface connector (Reference 7).

4.1.2.4 Low Voltage Differential Signaling

Low Voltage Differential Signaling (LVDS) is an open-standard alternative to TMDS. However, the primary LVDS standard from the American National Standards Institute (ANSI), namely ANSI/TIA/EIA-644-1995, Reference 23, is a generic physical layer standard. As such, it merely defines the low levels of a generic bit-pipe, and is not a digital video interface standard. There is an LVDS implementation standard, IEEE 1596.3, but this defines the physical layer of LVDS for Scalable Coherent Interface (SCI) applications. SCI is a scalable computer bus architecture for building large parallel-processing systems.

There are a number of LVDS display interface solutions from different silicon vendors. These FPD interfaces are not endorsed open standards, and consist of application notes, design support, and integrated circuit product lines. These include the FPD Link, Reference 24, from National Semiconductor and FlatLink™, Reference 25, from TI. LVDS technology has the ability to support high-resolution displays with the following 24-bit color resolutions: 1280 x 1024, 1600 x 1200, 1920 x 1080, and 2048 x 1536 (Reference 26).

4.1.2.5 American National Standards Institute Fibre Channel

Fibre Channel is an integrated set of ANSI standards that define new protocols for flexible information transfer (Reference 27). It is a common, efficient transport system supporting multiple protocols or raw data using native Fibre Channel guaranteed delivery services. Interoperability profiles define Fibre Channel uses for different protocols or applications (Reference 28). The physical interface is defined in a Fibre Channel standard known as FC-PH, Reference 8, including the bit pipe (copper or fiber), encoding, and framing. The upper-layer protocols provide application-specific mapping and protocols.

There are several approaches to using Fibre Channel in a FPD interface. The simplest, used by Battelle (Reference 2) for the CLADS head assembly test set, is to use the Fibre Channel FC-0 physical interface of Fibre Channel to provide a high-bandwidth bit-pipe. Raw video data is then transmitted on this bit-pipe in a dedicated point-to-point topology. Another approach is to build a high-speed Fibre Channel data network, and then embed video data packets into the network as high-priority unacknowledged packets (see 5.3 Network Display Interface). This is similar to the Internet streaming video formats such as RealVideo by RealNetworks, Reference 29.

An alternative is to define an upper-level mapping and add it to the Fibre Channel standards; there are two such efforts underway. The Fibre Channel Audio/Video (FC-AV) protocol is a transport mapping for video. The current draft, Reference 15, is Rev. 1.4, dated Sept. 17, 2000, available at ftp://ftp.t11.org/t11/pub/fc/av/00-252v3.pdf and contains 114 pages. The standard is currently in committee and is scheduled for release in June 2001. The FC-AV is fairly high-level, and is concerned mainly with digital television and MPEG packetization and transport, as well as future audio/video interfaces. It is primarily intended for broadcast applications to facilitate the 'digital studio' concept. The Fibre Channel Avionics Environment (FC-AE), Reference 16, consists of several extensions to the Fibre Channel standard for classes of service, login, connection management, and data security. The FC-AE is for avionics command, control, instrumentation, simulation, signal processing, and sensor/video data distribution. The current draft is dated April 9, 1996, and contains 74 pages of protocol enhancements and data mappings. As these extensions are approved, additional text is added to other Fibre Channel standards (such as FC-PH-3) for interoperable handling of the new protocols.

4.1.2.6 Gigabit Video Interface

Although not an open standard interface, the gigabit video interface (GVIF) from Sony represents a considerably different approach to FPD interfacing than either TMDS or LVDS. Most notably, GVIF uses a single link serial data transmission system on a single twinax cable. This allows very thin cables (up to ten meters) and miniature connectors to be used with GVIF (Reference 17). All synchronization and control data is embedded into the data stream, and a PLL is used in the receiver to regenerate the data clock, as shown in Figure 10.

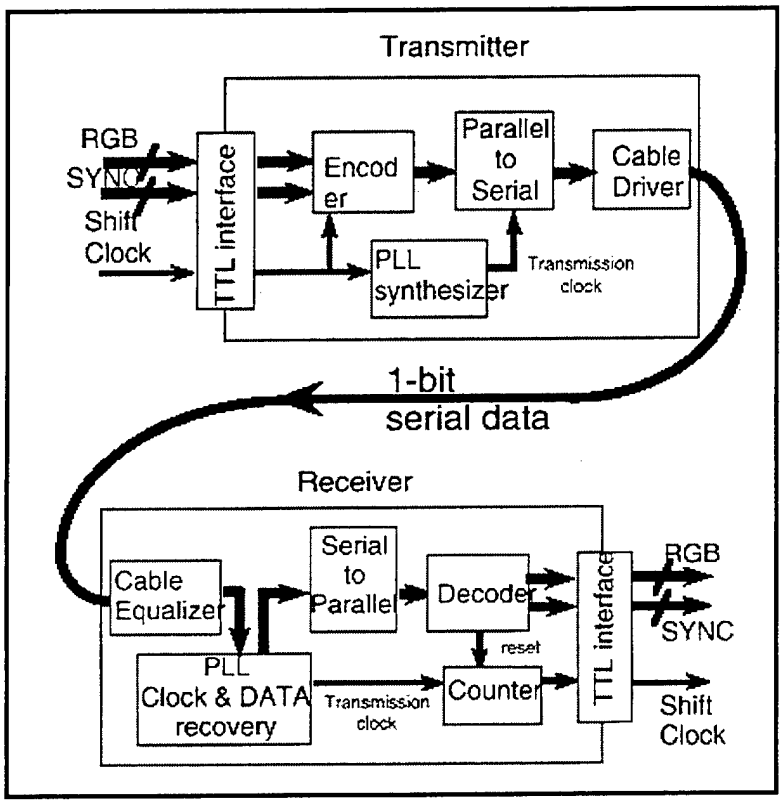


Figure 10. Gigabit Video Interface architecture (Reference 17).

There are, however, some limitations to this approach that make it difficult to apply with high resolution panels. GVIF is currently limited to only 512 colors (9-bit) at 1280 by 1024 resolution. Higher resolutions are not supported. The standard allows the use of a dual-link (two-pair cable) to provide 18-bit color at 1280 by 1024 resolution. This still falls short of the desired 24-bit support at resolutions beyond 1280 by 1024. A triple-link system (not referenced by the GVIF documents) might provide sufficient bandwidth, but at three twinax pairs, the GVIF no longer has a cabling advantage over TMDS or LVDS.

A more serious issue with GVIF may eliminate its consideration for ADI: it is a proprietary interface of Sony Corporation. The potential problems of basing ADI or any other interface on proprietary technology are alluded to in the following statement (taken from the GVIF chip set datasheet, Reference 30):

> "Sony reserves the right to change products and specifications without prior notice. This information does not convey any license by any implication or otherwise under any patents or other right. Application circuits shown, if any, are typical examples illustrating the operation of the devices. Sony cannot assume responsibility for any problems arising out of the use of these circuits."

Sony is currently the only company that can produce the necessary GVIF silicon, and they can change it or abandon it at any time. Similar statements can be found on the datasheets of the FPD Link and FlatLink™ LVDS solutions.

4.1.3 Enabling Technologies

The primary interface technologies are already discussed in above. The most relevant interface technologies, TMDS and Fibre Channel differential positive emitter-coupled logic (PECL), are both used in ADI. The focus here is the most cost-effective methods of incorporating these into ADI.

The interface issues center on the graphics accelerator integrated circuits (ICs). The current and emerging COTS accelerator ICs provide significant capability for both fast workstation graphics and advanced 3D capability. These accelerator ICs offer an excellent opportunity for military systems to leverage commercial advances in computer graphics. These ICs may or may not include internal RAMDACs for CRT compatibility. Since most proposed commercial interfaces use TMDS, it is reasonable to expect that future COTS graphics accelerator ICs will integrate TMDS drivers and provide direct TMDS outputs, just as current ICs integrate RAMDACs and provide direct analog outputs. If this is the case, all ADI solutions will have to deal with TMDS digital video at the graphics generator end of the interface. The newest graphics accelerator IC from ATI includes integrated TMDS drivers for no-cost FPD support (Reference 14).

Most new and emerging COTS flat panel monitors include DVI input capability. The necessary TMDS receivers have been integrated into LCD monitor controller ICs for low-cost DVI support.

TMDS drivers and receivers are also available separately from industry vendors. These are used to implement the standard DVI interface for ADI. The TMDS ICs include all the necessary logic, timing, synchronization, and control functions integrated into a single IC.

Fibre Channel transmitters and receivers are available from several vendors that implement the entire physical layer interface in a single IC. These are used to build a high-bandwidth data pipe using legacy cable (coax, miniature coax, twinax, twisted-pair, shielded twisted-pair, and optical fiber). The Fibre Channel ICs include all the necessary coding, clock recovery, synchronization, and control functions integrated into a single IC.

4.2 Demonstration System

The demonstration system defined in section 2.2 was designed. This includes procurement of the COTS components and design of the ADI transmitter and repeater. The demonstration system is shown in Figure 11 below.

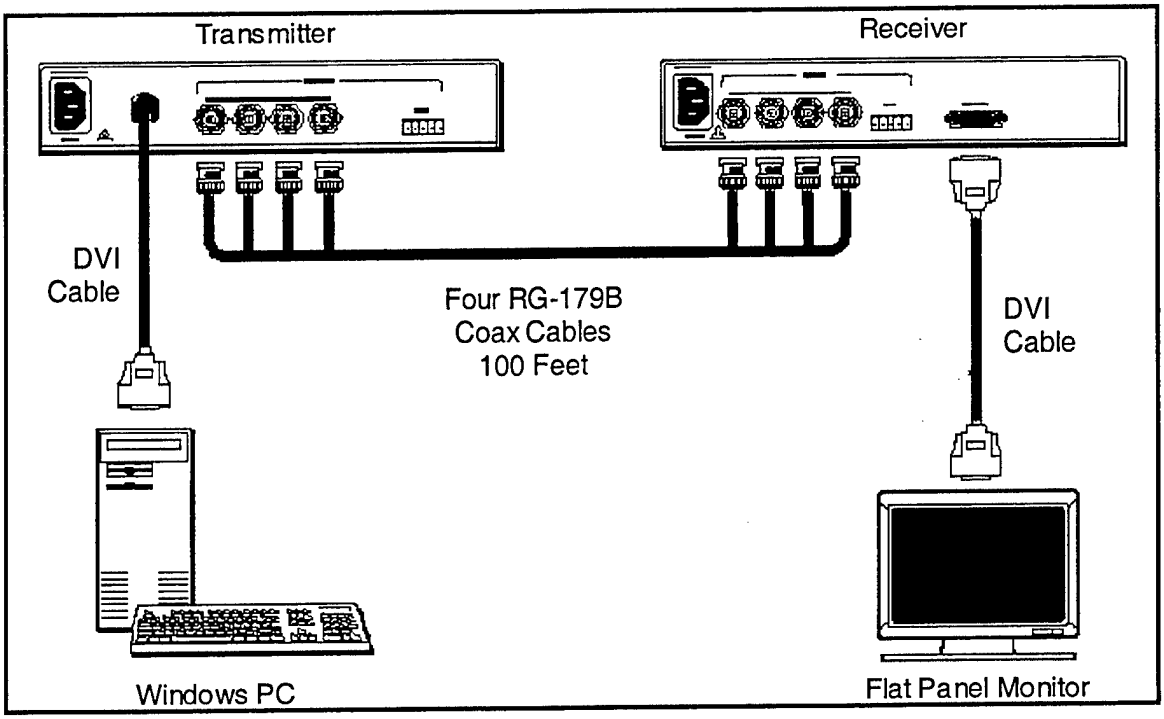


Figure 11. Demonstration system.

4.2.1 Desktop Computer

The graphics generator is a COTS Microsoft Windows® PC with a DVI output graphics card. The COTS PC is a custom-built model from Multiwave Technology, 17901 East Ajax Circle, Industry, California, 91748. The PC configuration is shown in Table 4.

Table 4. Personal computer configuration.

Model	Item	Description (company name, designation)
C830-59	Motherboard & Central Processing Unit	ABIT KT7A Motherboard with AMD Thunderbird 1 GHz & Cooling Fan
A06275	Operating System	Microsoft Windows® 98 2nd Edition
A02610	Case & Power Supply	Mwave CI-6606/4 Middle Tower with 300 W ATX Power Supply
A03470	Memory	Multiwave 32X64 PC133 256MB SDRAM DIMM
130570	Graphics Card	ATI Radeon All-In-Wonder 32MB AGP NTSC
081789	Hard Drive	Western Digital 40.0GB WD400BBRTL EIDE Ultra-ATA/100 7200RPM
A00803	Compact Disk (read only)	Aopen CD950E 50x EIDE Internal
A00695	Floppy Drive	Sony 1.44MB
682373	Network Card	3Com 10BT/100BTX Fast Etherlink XL 3C905BTXNM PCI
000044	Mouse	Microsoft Intellimouse Optical PS2/USB
511815	Keyboard	Microsoft Natural Elite PS/2 Keyboard

* ABIT, AMD, Microsoft, Mwave, Multiwave, ATI, Western Digital, Aopen, Sony, and 3Com are names of companies. Designations are provided by the companies to identify their product model; refer to company literature for explanations.

4.2.2 Flat Panel Display

The display device is a COTS desktop AMLCD monitor. The monitor is model number VG181 from ViewSonic Corporation, 381 Brea Canyon Road, Walnut, California, 91789. The VG181 has 1280 by 1024 native resolution and an 18 inch diagonal viewing area. The VG181 has both analog RGB and DVI inputs. Full specifications are shown in Table 5.

4.2.3 Legacy Cable

The legacy cable installation is simulated using a 100 ft spool of RG-179B coax, with four separate cable strands. This cable is available from Alpha Wire Company, 711 Lidgerwood Avenue, Elizabeth, New Jersey, 07207. The cable specifications are listed in Table 6. The cables are terminated with standard baby "N" connectors (BNCs) at each end. The BNCs are used on the IEEE 802.3 (Ethernet) standard type RG58 coaxial networks.

Table 5. Flat panel display specifications.

Parameter	Performance
Active Area Size	18.1 in. diagonal (4:3 aspect)
True Resolution	1280 x 1024 pixels
Brightness	235 nit
Contrast Ratio	300:1
Viewing Angle	160° horizontal 160° vertical
Glass Surface	Anti-glare
Video Inputs	RGB (analog) & DVI-V (digital)
Bandwidth	135 MHz
Sync Inputs	Separate TTL* Composite sync Sync on green
Sync Frequency Range	30-82 kHz horizontal 50-75 Hz vertical
Connectors	15-pin mini D-sub (analog) DVI-V (digital)
Power	90-264 VAC, 50-60 Hz, 70 W
Dimensions	18.1 x 18 x 9.4 in.
Weight	22 lbs

* TTL: transistor-to-transistor logic

Table 6. Cable Specifications.

Parameter	Performance
Part Number	9179B
Type	RG-179 B/U 75Ω coaxial
Inner Conductor	30 AWG* stranded silver coated copper coated steel
Dielectric	TFE** Teflon
Shield	93% braid silver coated copper
Jacket	Fluorinated Ethylene-Propylene
Velocity of Propagation	70%
Capacitance	19.6 pF/ft
Attenuation	100 MHz: 9.8 db/100 ft 200 MHz: 12.7 db/100 ft 400 MHz: 15.8 db/100 ft 1000 MHz: 25.0 db/100 ft

* AWG: American wire gage

** TFE: Tetraflourethylene (Teflon)

4.2.4 Advanced Display Interface Transmitter

The ADI transmitter accepts DVI and outputs Fibre Channel data to the coax cables. The transmitter block diagram is shown in Figure 12.

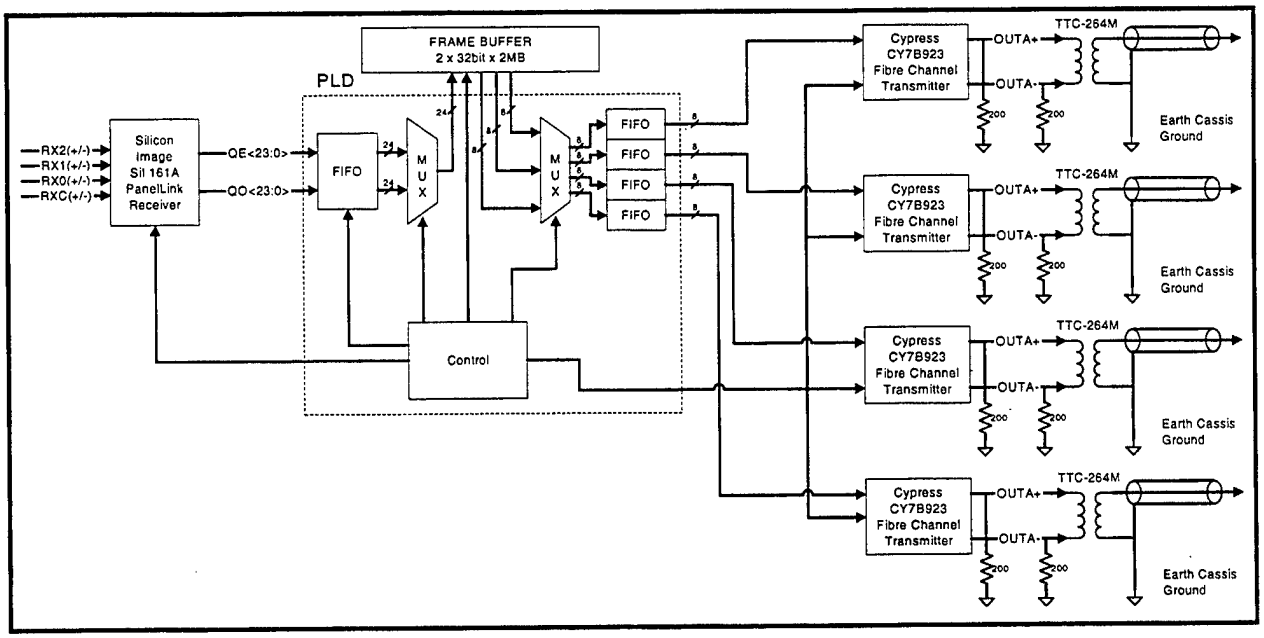


Figure 12. Advanced Display Interface transmitter.

The transmitter schematic diagram is given in Appendix A. The layout of the transmitter circuit board is shown in Appendix C. The Very High Level Description Language (VHDL) source code for the transmitter logic is provided in Appendix E.

4.2.5 Advanced Display Interface Receiver

The ADI receiver accepts Fibre Channel data from the coax cables and outputs DVI to the display. The receiver block diagram is shown in Figure 13.

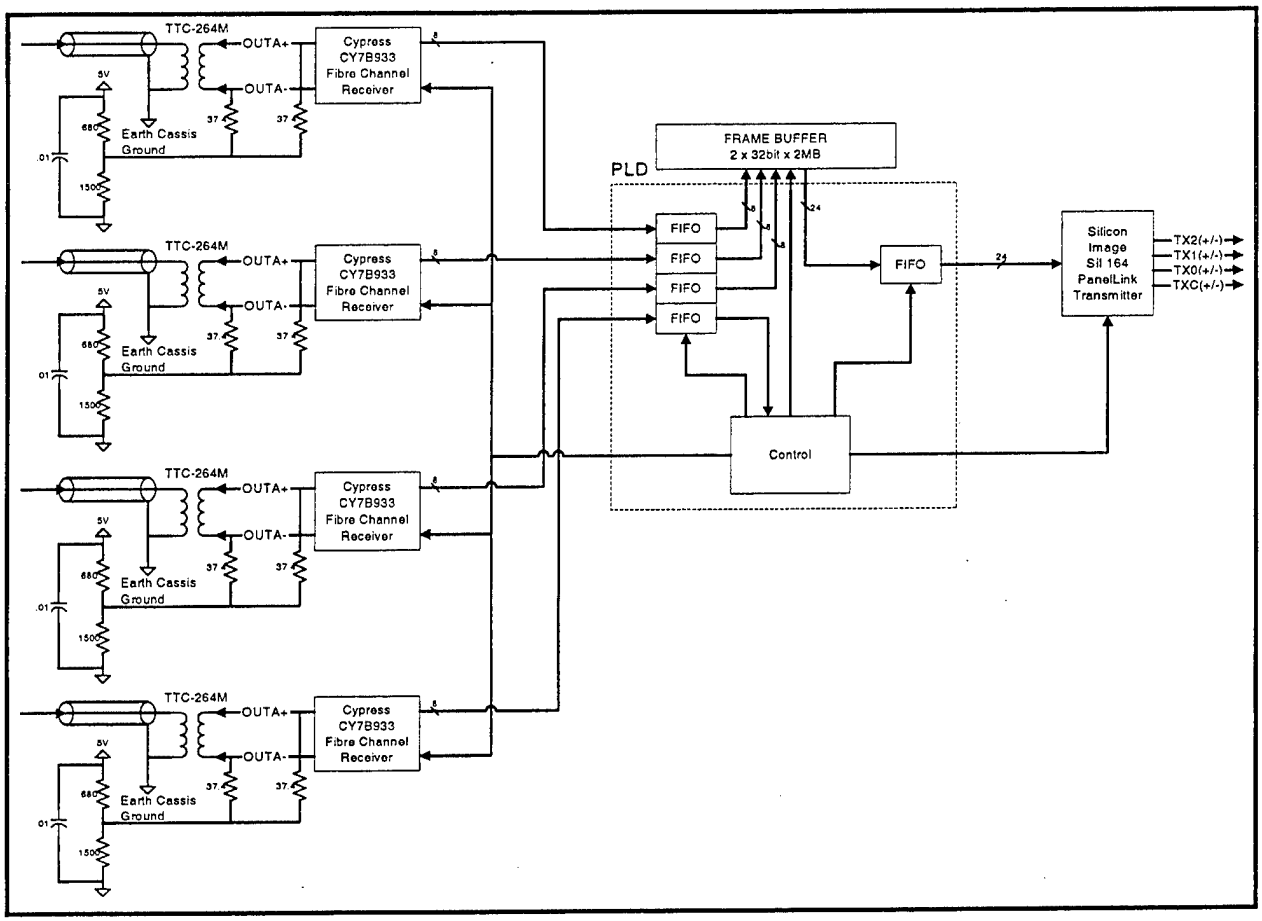


Figure 13. Advanced Digital Interface receiver.

The receiver schematic diagram is given in Appendix B. The layout of the transmitter circuit board is shown in Appendix D. The VHDL source code for the transmitter logic is provided in Appendix F.

5. CONCLUSIONS

From the technology assessment, the most promising commercial interface today is DVI. This interface can be adapted for use by military display systems provided some shortcomings are addressed. One of the most significant limitations of DVI is cable type and length. Standard DVI must use a specific type of cable not found in legacy military applications and is limited to ten meter cable length. This limitation is overcome using an ADI transmitter/receiver pair to interface with legacy cable.

The legacy cable will not support the highest data rates possible with DVI. Several methods are used in combination to reduce the required data rate so that existing video cabling can be used. These include bit-depth reduction from 24-bits to 16-bits and frame rate reduction from 30 Hz interlaced to 15 Hz non-interlaced. Also, the legacy cable bandwidth will ultimately limit the maximum resolution possible with a usable frame rate.

A prototype system has been designed with an ADI transmitter/receiver pair. The transmitter converts standard DVI from a COTS graphics card to Fibre Channel based data for transmission on coax cables. The receiver converts the Fibre Channel data back to standard DVI for a COTS flat panel display. The transmitter and receiver utilize both bit-depth reduction and frame rate reduction to transmit digital video on one to four coax cables.

The ADI transmitter and receiver circuit boards are small size and low power. They can be produced at relatively low cost.

6. RECOMMENDATIONS

6.1 Interface Definition

The next effort is to complete the interface definition to create an industry standard open systems interface. The result of this effort should be a formal specification of the complete ADI interface. This would include a graphics card specification for graphics modes, software interface, digital data formats, and connectors. Also included would be the display specifications for supported graphics modes, digital data formats, and connectors. The complete ADI standard should include cable specifications and open protocol definitions in addition to the above.

Once the ADI standard is complete, commercialization of ADI should be encouraged through standardization. The ADI standard should be proposed to commercial standards organizations as an open standard or extension to existing standard. There may be an ADI extension to DVI or a Fibre Channel upper layer protocol for ADI as a Class 4 (fractional bandwidth) or Class 6 (unidirectional broadcast) Fibre Channel service.

6.2 Advanced Display Interface Adoption

An important step towards adopting ADI in military systems is to demonstrate the interface in a military application. Several efforts could result in a complete demonstration system that can be refined into a final interface solution. First, an ADI graphics card should be developed. This digital graphics card provides ADI video from a workstation computer. The ADI graphics card may consist of a COTS digital graphics card and a format converter. Next, an ADI receiver should be developed and integrated with a military FPD. An example is a CLADS AVIM designed to receive ADI video and display it on a CLADS head assembly.

Once the prototype ADI hardware is developed, it should be demonstrated in a military system. For example, the ADI system could be demonstrated on the E-3 AWACS with CLADS. Any high-resolution military FPD application is a good candidate for an ADI demonstration and eventual integration. The lessons-learned during the development efforts and demonstrations would then lead to refinements of both the ADI specification and the ADI system design.

6.3 Network Display Interface

Another important variant of the ADI is the Network Display Interface (NDI) for network display applications. The NDI is a digital data network version of ADI, where video data is packetized and transported using a COTS standard data network. With the NDI, graphics generators are plugged into a data network and driver software is used to create NDI packets that are injected into the network. Each NDI display also plugs into the same network and receives, decodes, and displays the video data. The NDI displays may be intelligent, allowing several windows to display multiple NDI streams from several sources.

To develop the NDI, first an assessment needs to be conducted to define the NDI architecture. The assessment should include potential military applications and enabling technologies. This should include analysis of Internet streaming video formats and video compression techniques. Next, a NDI specification should be developed. The result is an industry standard open systems display video interface for networks. The standard could be supported by several COTS network fabrics, including Fibre Channel and Ethernet. Finally, a prototype NDI system should be developed. The goal of the prototype system is to demonstrate packetized display video over a COTS data network.

7. LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ADC	Analog-to-digital converter
ADI	Advanced display interface
AFRL	Air Force Research Laboratory
AMLCD	Active matrix liquid crystal display
AVIM	Application Video Interface Module (part of CLADS)
ANSI	American National Standards Institute
AWACS	Airborne Warning and Control System
AWG	American Wire Gauge
BAA	Broad Agency Announcement
BIT	Built-in test
BMI	Battelle Memorial Institute
BNC	Baby "N" connector (used on Ethernet type RG58 coaxial networks)
bps	bits per second
CLADS	Common Large Area Display Set
Coax	Coaxial (cable)
COTS	Commercial-off-the-shelf
CRT	Cathode ray tube
DARPA	Defense Advanced Research Projects Agency
DDC2	Display Data Channel 2 (VESA standard data interface to display)
DDI	Digital display interface
DDWG	Digital Display Working Group
DFP	Digital Flat Panel (VESA standard)
DLP™	Digital Light Processing (TI trademark for projection light engines using DMDs)
DMD	Digital micromirror device
DPMS	Display Power Management Standard (VESA standard)
DSTN	Dual scan twisted nematic (LCD)
DVI	Digital Video Interface (DDWG standard)
DVI-V	Designation for DVI digital connector
EDID	Extended Display Identification Data (VESA standard)
EMI	Electromagnetic interference
Ethernet	Nickname for the IEEE 802.3 standard for access/distribution layers of a data communications system; 100 Mbps to 100m; 1 Gbps to 10 km; 10 Gbps to 40 km
FC	Fibre Channel
FC-AE	Fibre Channel Avionics Environment
FC-AV	Fibre Channel Audio Video (for TV and MPEG)
FC-PH	Fibre Channel, Physical Interface
FCC	Federal Communications Commission
Fibre Channel	Nickname for the set of ANSI standards for high-speed data transfer among workstations, mainframes, and display devices for distances to 10 km
FIFO	First-in first-out (signal protocol)
Firewire	Nickname for the IEEE1394 standard for bus forming digital network backbone; IEEE1394b: 800 Mbps in Versatile Home Network; potential: 1.6 Gbps to 12 m; IDB-1394: 24.8Mbps in autos for Media Oriented Transport Consortium (Most)
FPD	Flat panel display

fps	frames-per-second
GB	Gigabyte, 10^9 bytes (1 byte is typically equal to 8 bits)
Gbps	Gigabits per second
GVIF	Gigabit video interface (Sony standard)
HDL	High Level Description Language
HDS	High Definition Systems
HDTV	High Definition Television (720 to 1080 lines, 1024 to 1920 spots/line, interlaced or progressive scan)
HSDDI	High Speed Digital Display Interface (part of CLADS)
IC	Integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IP	Intellectual property
LCD	Liquid crystal display
LDWG	Large Display Work Group
LVDS	Low Voltage Differential Signaling
MB	Megabyte, 10^6 bytes (1 byte is typically equal to 8 bits)
MDR	Mini-D ribbon (20-pin connector)
MPEG	Motion Pictures Experts Group
NDI	Network Display Interface
NIST	National Institute for Standards and Technology, US Department of Commerce
nit	Candela per meter^2 (standard unit of luminance)
NTSC	National Television Standards Committee U.S. TV broadcast standard (525 lines, 336 spots/line, interlace scan)
P&D	Plug and Display (VESA standard, 30-pin connector)
PCI	Personal computer interface
PECL	Positive emitter-coupled logic (used in Fibre Channel)
PLD	Programmable logic device
PLL	Phase locked loop
QXGA	Quad-XGA (3,145,728 pixels in 2048 x 1536 format)
QUXGA	quad-UXGA (7,680,000 pixels in 3,200 x 2,400 format)
QUXGA-W	QUXGA-wide (9,216,000 pixels in 3840 x 2400 format)
RAMDAC	Random access memory digital-to-analog converter
RG-179B	Miniature coax cable (75 Ω), used for AWACS crewstations (legacy equipment)
RGB	Red, Green, Blue
RMS	Root mean square
RPM	Revolutions per minute
RTCA/DO	Radio Technical Commission for Aeronautics / DOcument
SCI	Scalable Coherent Interface
SID	Society for Information Display
SPIE	International Society for Optical Engineering (previously known as Society of Photo-Optical Instrumentation Engineers)
SVGA	Super VGA (480,000 pixels in 800 x 600 format)
SXGA	Super-extended Graphic Array (standard involving 1280 x 1024 pixel resolution)
SYNC	Synchronization (signal)
TFE	Tetraflourethylene, C_2F_4 (polymerization of this monomer produces Teflon)

TFT	Thin-film transistor
TI	Texas Instruments
TMDS	Transition Minimized Differential Signaling
TTL	Transister-to-transister logic
TV	Television
twinax	twin axial (cable)
USB	Universal Serial Bus
UXGA	Ultra-extended Graphic Array (1,920,000 pixels in 1600 x 1200 format)
VAC	Volts, alternating current
VESA	Video Electronics Standards Association (VESA)
VGA	Video Graphic Array (307,200 pixels in 640x480 format with 15-pin connector)
VHDL	Very High-Level Description Language
WR-ALC	Warner-Robins Air Logistics Center
WSXGA	Wide-SXGA (1,474,560 pixels in 1440 x 1024 format)
XGA	eXtended Graphics Array (786,432 pixels in 1024 x 768 format)

8. REFERENCES

1. Ronald L. Gorenflo and David J. Hermann, "21 inch technology independent common display set (CLADS) design for rugged workstation applications," in *Cockpit Displays IV: Flat Panel Displays for Defense Applications*, Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 3057, 428-439 (1997).

2. David J. Hermann and Ronald L. Gorenflo, "Analog and digital interfaces for the common large area display set," in *Cockpit Displays IV: Flat Panel Displays for Defense Applications*, Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 3057, 446-456 (1997).

3. *High Speed Digital Display Interface (HSDDI) Interface Control Document*, Battelle, Document No. 3310030015, Columbus, Ohio, 2 April 1998. Available to DoD Agencies and DoD Contractors via request to WR-ALC/LYLCB, Robins AFB GA 31098-1638.

4. D. Hopper, F. Meyer, "Advanced aerospace display interfaces," in *Cockpit Displays V: Displays for Defense Applications*, Darrel G. Hopper, Editor, Proceedings of SPIE Volume 3363, 448-459 (1998).

5. VESA Plug and Display (P&D™) Standard, Video Electronics Standards Association (VESA), Version 1, 11 June 1997, http://www.vesa.org.

6. VESA Digital Flat Panel (DFP) Standard, Video Electronics Standards Association (VESA), Version 1, 14 February 1999, http://www.vesa.org.

7. Digital Visual Interface (DVI), Digital Display Working Group, Revision 1.0, 2 April 1999, http://www.ddwg.org.

8. Fibre Channel Physical and Signaling Interface – 2 (FC-PH-2), Computer & Business Equipment Manufacturers Association, American National Standard X3T11 Committee, Project 901-D, Draft Revision 7.4, 10 September 1996, http://www.fibrechannel.com.

9. VESA Display Data Channel (DDC™) Standard, Video Electronics Standards Association, Version 3, 15 December 1997, http://www.vesa.org.

10. (a) D.D. Desjardins and D.G. Hopper, *Military Display Market: Second Comprehensive Edition*, Technical Report AFRL-HE-WP-TR-1999-0211 (August 1999), 434 pp. Available to Government Agencies and their Contractors via request to AFRL/HECV (Dr. Hopper), 2255 H Street, Room 300, Wright-Patterson AFB OH 45433-7022; (b) Darrel G. Hopper and Daniel D. Desjardins, "Military display market: second comprehensive edition," in *Cockpit Displays VII: Displays for Defense Applications*, Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 4022, 19-28 (2000); (c) Daniel D. Desjardins and Darrel G. Hopper, "Military display market segment: aerospace cockpits," in *Cockpit Displays VIII: Displays for Defense Applications*, Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 4362, paper 4 (2001).

11. David J. Hermann and Ronald L. Gorenflo, "Digital interface for high resolution displays," in *Cockpit Displays VI: Displays for Defense Applications,* Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 3690, 317-328 (1999).

12. (a) Darrel G. Hopper, William K. Dolezal, Keith Schu, and John W. Liccione, *Draft Standard for Color Active Matrix Liquid Crystal Displays (AMLCDs) in U.S. Military Aircraft,* Wright Laboratory Technical Report WL-TR-93-1177, 60 pp (June 1994); available from the Defense Technical Information Center (DTIC), Springfield VA, DTIC Accession No. AD-A282950. Wright Laboratory became part of the Air Force Research Laboratory in 1997. Report is also available from AFRL/HECV (Dr. Hopper), WPAFB OH 45433-7022;
(b) Darrel G. Hopper, William K. Dolezal, Keith Schu, and John W. Liccione, "Draft Standard for Color AMLCDs in U.S.Military Aircraft," in *Cockpit Displays*, Darrel G. Hopper Editor, Procceedings of SPIE Vol. 2219, 230-238 (1994). Summary of the technical report.

13. Environmental Conditions and Test Procedures for Airborne Equipment, Radio Technical Commission for Aeronautics, December 1989, RTCA/DO-160C.

14. "ATI, the leader in digital flat panel graphic accelerators, announces support for Intel's Digital Visual Interface," ATI Technologies, Inc. Press Release, Toronto, 8 April 1999, http://www.atitech.com/ca_us/corporate/press/1999/4183.html.

15. Fibre Channel – Audio Video (FC-AV), National Committee for Information Technology Standardization, American National Standard T11 Committee, Project 1237-D. Rev. 1.4, dated Sept. 17, 2000, is available at ftp://ftp.t11.org/t11/pub/fc/av/00-252v3.pdf and contains 114 pages. http://www.fibrechannel.com.

16. Fibre Channel Avionics Environment (FC-AE), X3T11 Avionics Environment Working Group, Project 2009-D, Draft Revision 0.4, 9 April 1996, http://www.fibrechannel.com.

17. H. Kikuchi, T. Fukuzaki, R. Tamaki, and T. Takeshita, "Gigabit Video Interface: A Fully Serialized Data Transmission System for Digital Moving Pictures," International Conference on Consumer Electronics (ICCE), 1998, http://www.world.sony.com/Electronics/SC-HP/N_Techno/GVIF/PDF/icce98d.pdf.

18. "VESA Mission Statement," Video Electronics Standards Association, http://www.vesa.org/mission.html.

19. "Compaq PC Technologies: Digital Flat Panel (DFP) Port," Compaq Computer Corporation, 1998, http://www.compaq.com/athome/pc_technologies/pcdemo_dfp.html.

20. "VESA Adopts Digital Flat Panel (DFP) Standard," Video Electronics Standards Association (VESA) Press Release, 29 March 1999, http://www.vesa.org/news032999.html.

21. "Intel, Compaq, Dell, Fujitsu, Hewlett Packard, IBM, Microsoft, NEC, Silicon Image Form Digital Display Working Group To Define Digital Connectivity Specification," Intel Developer Forum, Palm Springs, CA, September 17, 1998, http://www.intel.com/pressroom/archive/releases/CN91798b.HTM.

22. "DVI Frequently Asked Questions," Digital Display Working Group, http://www.ddwg.org/faq.html.

23. Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits, Telecommunications Industry Association, ANSI/TIA/EIA-644-1995, 25 March 1996, http://www.tiaonline.org.

24. S. Poniatowski, "An Introduction to FPD Link," National Semiconductor Application Note 1032, July 1998, http://www.national.com/an/AN/AN-1032.pdf.

25. K. Gingerich, "FlatLink™ Data Transmission System Design Overview," Mixed-Signal Products, Texas Instruments, Inc., 1997, http://www-s.ti.com/sc/psheets/slla012/slla012.pdf.

26. J. Goldi, and G. Nicholson, "A case for low voltage differential signaling as the ubiquitous interconnect technology," Electronic Systems, Vol. 38, No. 3, p. 42, March 1999, http://www.estd.com.

27. What is Fibre Channel?, Ancot Corporation, 3rd Ed., October 1996, ISBN 0-9637439-1-0, http://www.ancot.com.

28. Fibre Channel: Connection to the future, Fibre Channel Association, 2nd Ed., 1998, ISBN 1-878707-45-0, http://www.fibrechannel.com.

29. V.Thomas, "IP Multicast in RealSystem G2," RealNetworks, Incorporated, White Paper, 15 January 1998, http://docs.real.com/docs/g2multicast.pdf.

30. "Gigabit Video Interface 24-bit Color Chip Set," CXB1454R/55R/56R Preliminary Datasheet, Sony Semiconductor, 3 February 1999, http://www.world.sony.com/Electronics/SC-HP/N_Techno/GVIF/PDF/GVIF24E3.pdf.

31. Steven J. Hoener, John Flint, and Alan R. Jacobsen, "Common display performance requirements for military and commercial aircraft product lines," in *Cockpit Displays VIII: Displays for Defense Applications,* Darrel G. Hopper, Editor, Proceedings of SPIE Vol. 4362, paper 14 (2001).

32. *Measurements of Electromagnetic Interference Characteristics,* Military Standard number MIL-STD-462D (11 January 1993).

APPENDIX A

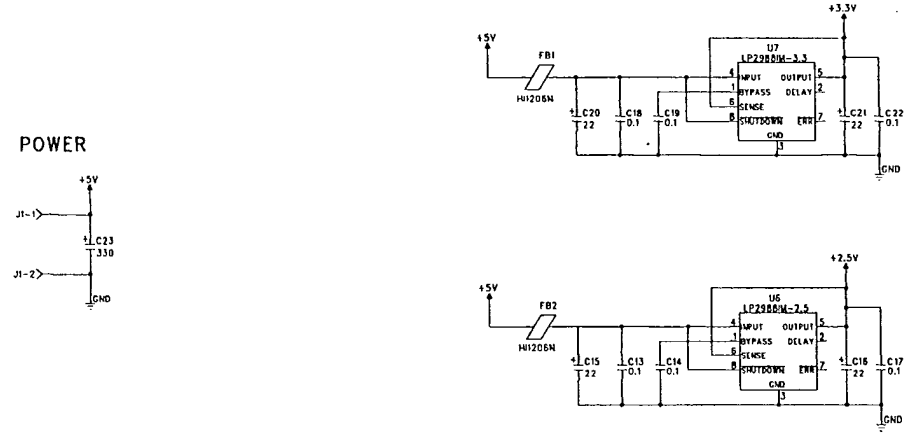
TRANSMITTER SCHEMATIC

The following pages are the schematic diagram for the ADI transmitter designed for the demonstration system.

NOTES:

1. INTERPRET DRAWING PER MIL-STD-100.
2. THIS SCHEMATIC IS USED IN CONJUNCTION WITH CIRCUIT CARD ASSEMBLY.
3. ALL RESISTORS ARE IN OHMS, 1/8 W, 1% UNLESS OTHERWISE NOTED.
4. ALL CAPACITORS ARE IN MICROFARADS UNLESS OTHERWISE NOTED.
5. LAST COMPONENT USED:

POWER CONVERSION



POWER

REVISIONS

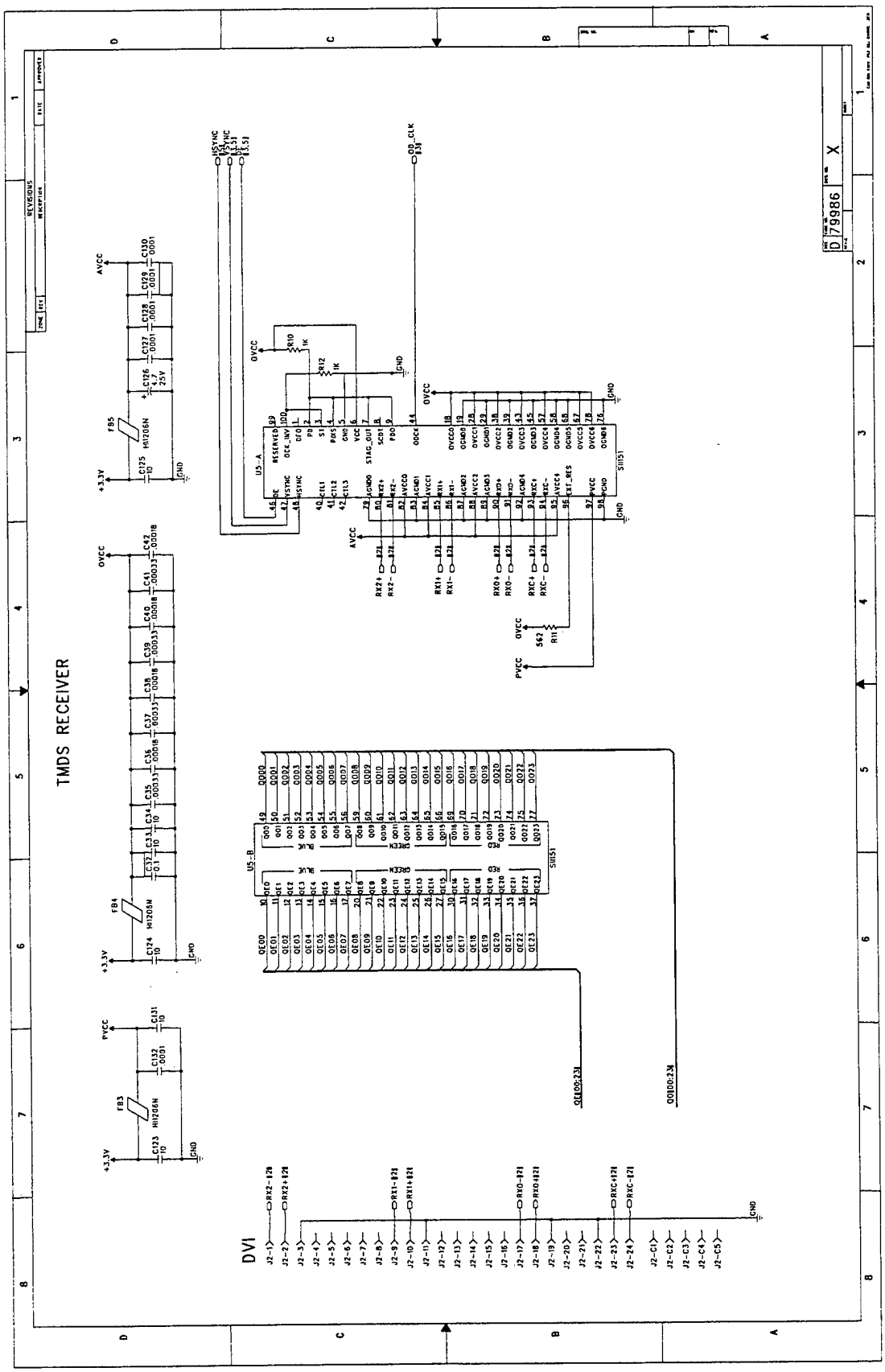
ZONE	REV	DESCRIPTION	DATE	APPROVED

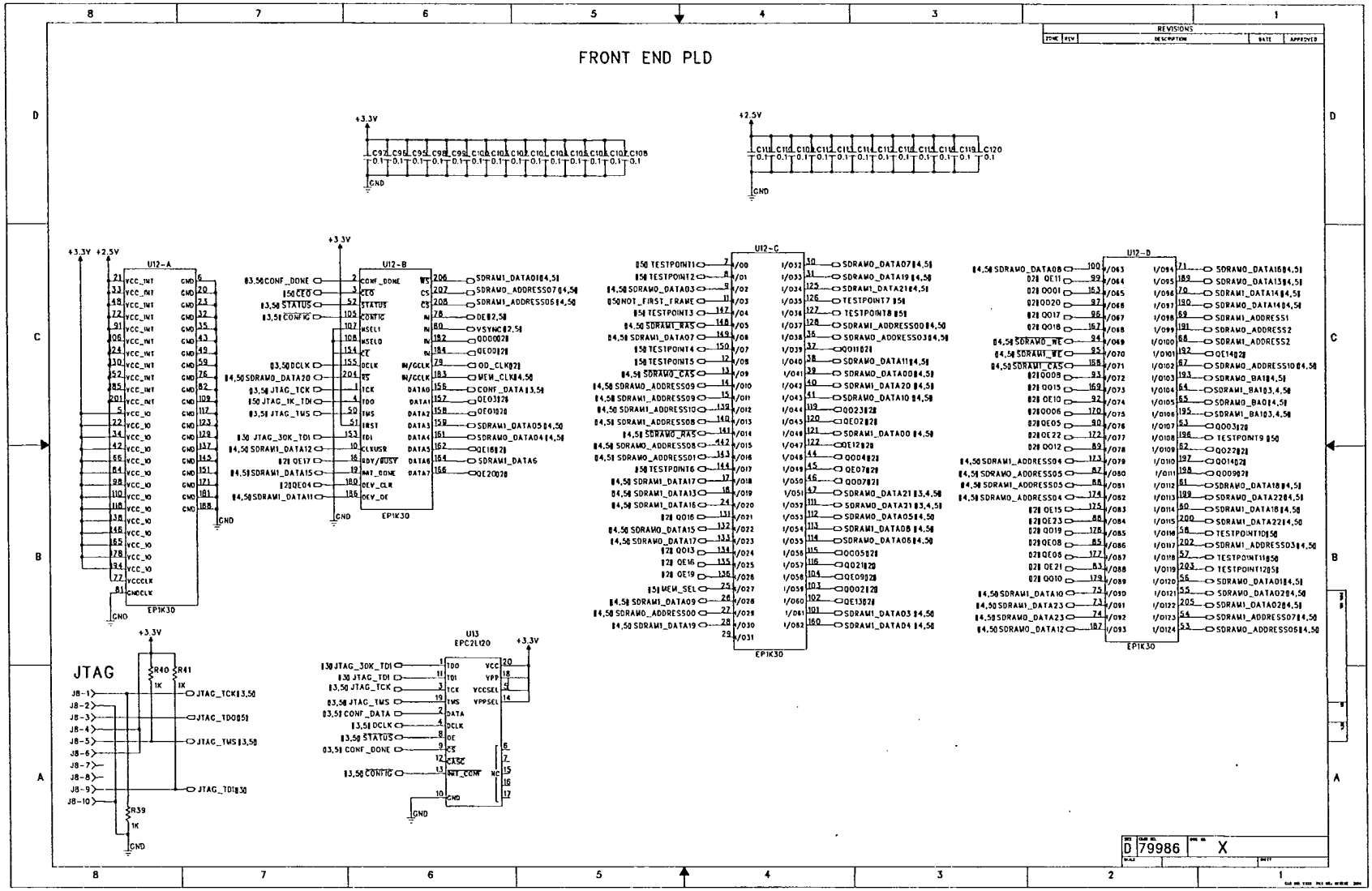
PARTS LIST

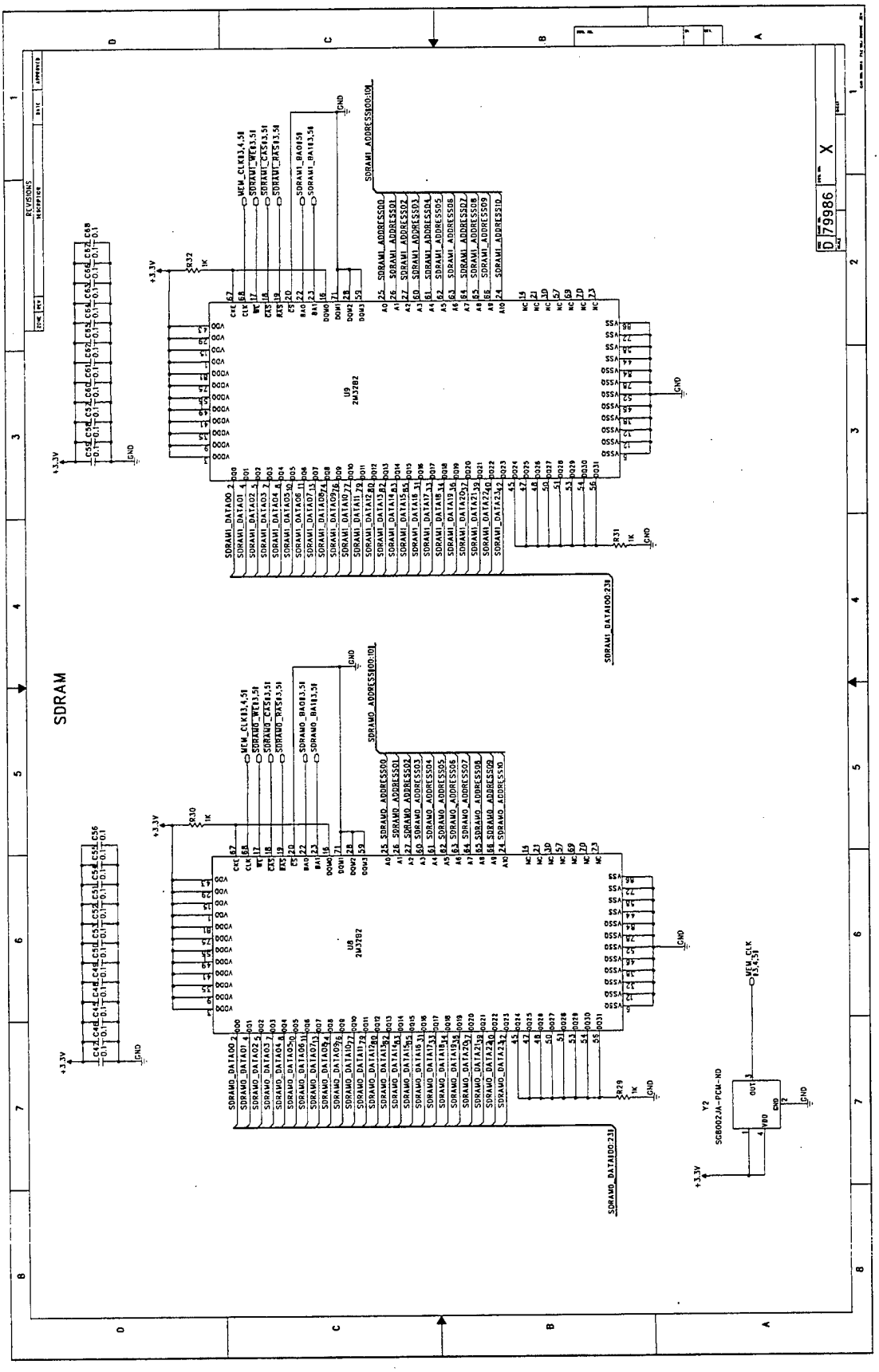
ADI
TX_MODULE

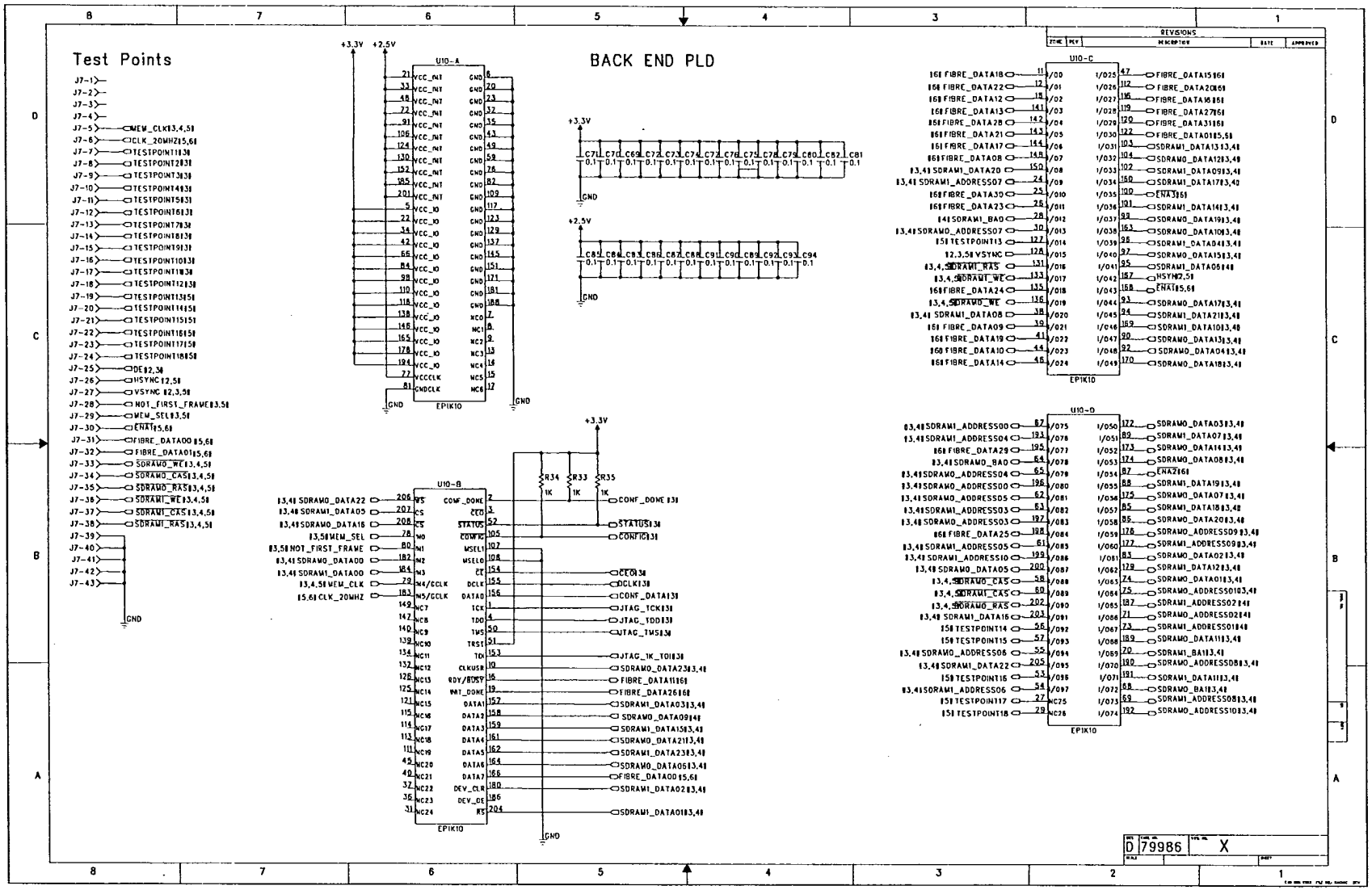
D 79986 X

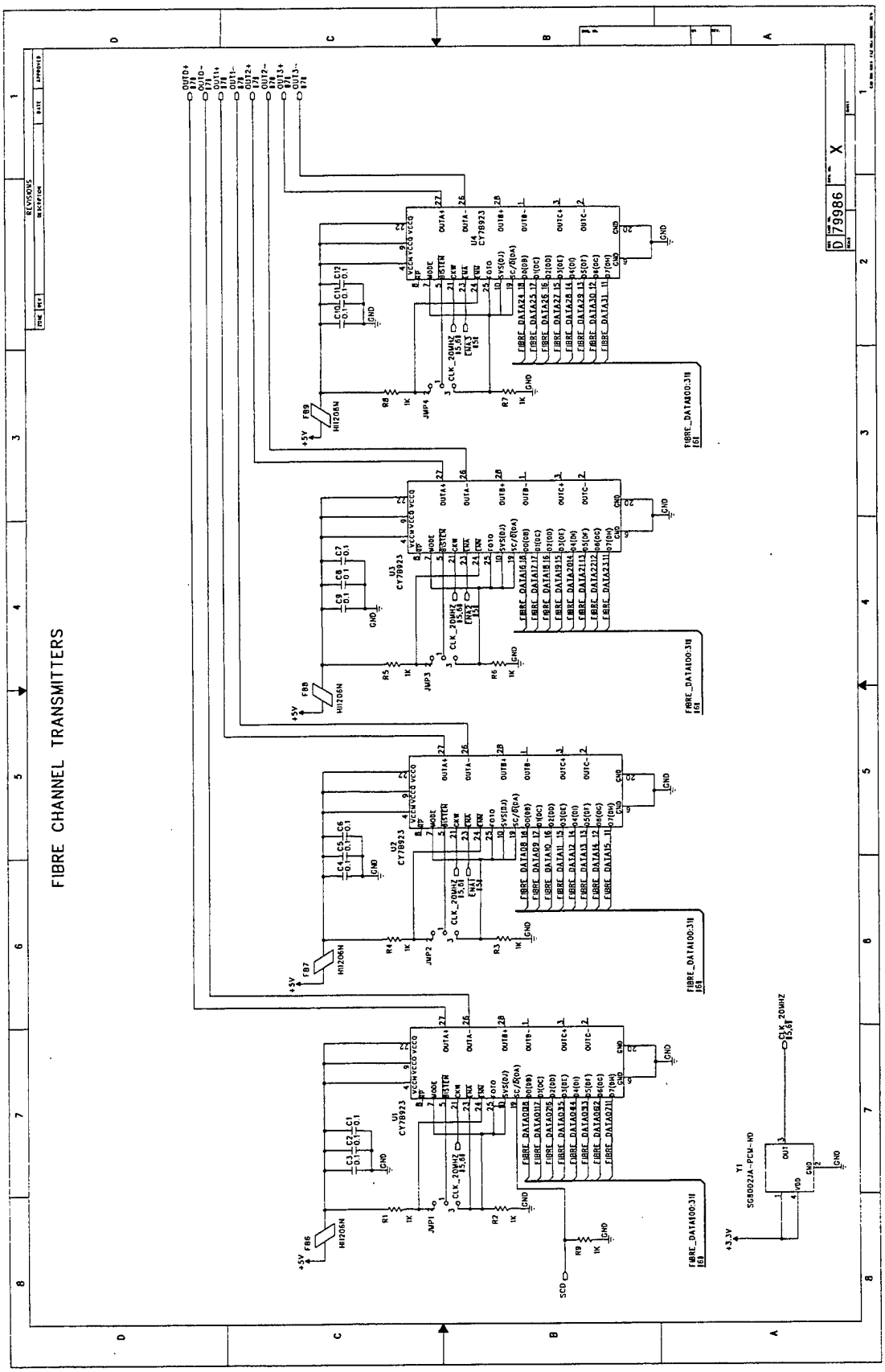
SHEET 1 OF

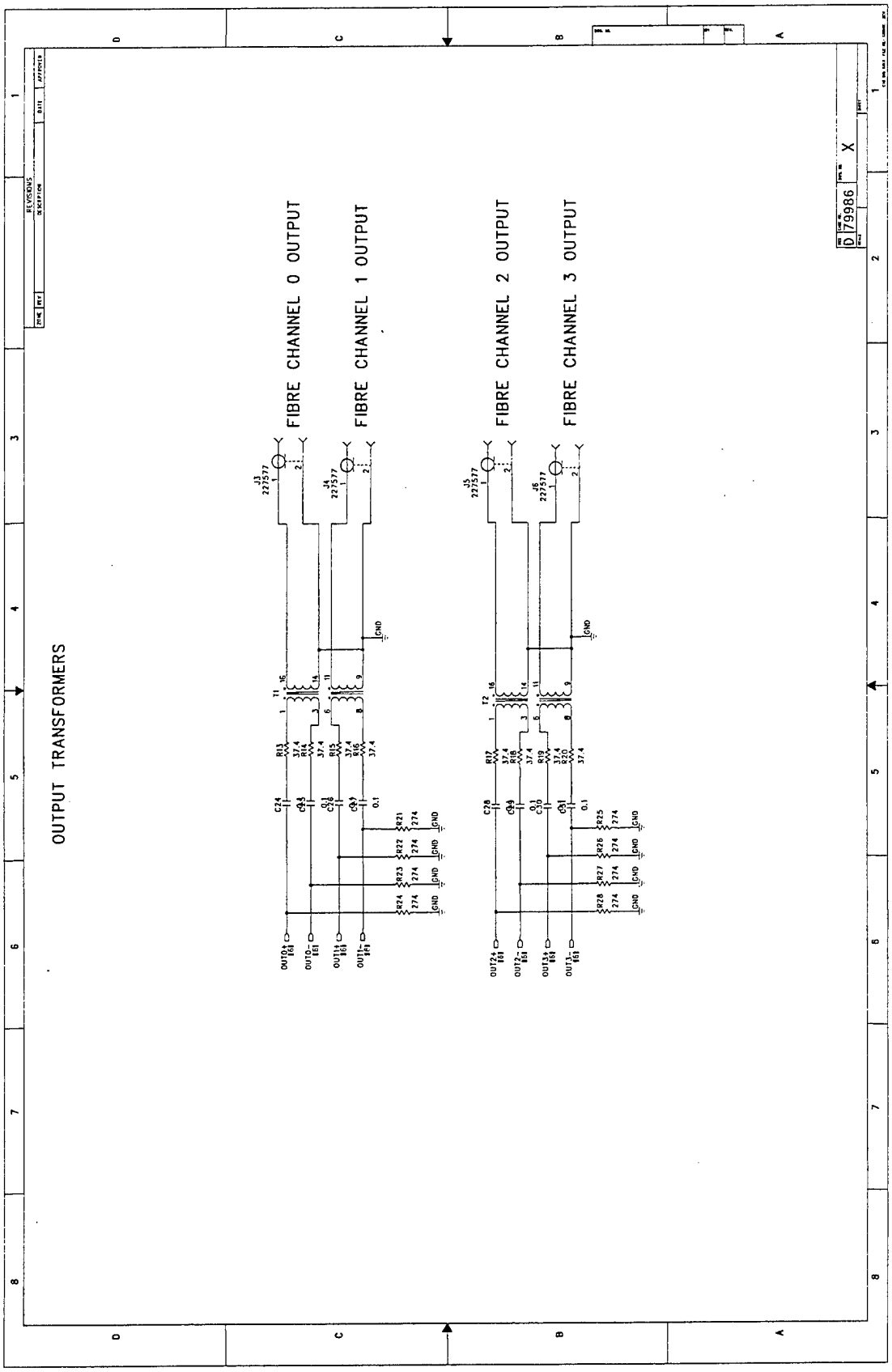

TMDS RECEIVER
DVI
U5-A
U5-B
SI1151
D79986
X


FRONT END PLD
REVISIONS
ZONE REV DESCRIPTION DATE APPROVED
+3.3V
+2.5V
GND
U12-A
U12-B
U12-C
U12-D
EP1K30
U13
EPC2LI20
JTAG
R40 1K
R41 1K
R39 1K
JB-1
JB-2
JB-3
JB-4
JB-5
JB-6
JB-7
JB-8
JB-9
JB-10
JTAG_TCK[3,5]
JTAG_TDO[5]
JTAG_TMS[3,5]
JTAG_TDI[3]
D 79986
X


SDRAM
U9
2M32B2
U8
2M32B2
Y2
SG8002JA-PCM-ND
D 79986 X


Test Points
BACK END PLD
U10-A
U10-B
U10-C
U10-D
EP1K10
REVISIONS
D 79986 X


FIBRE CHANNEL TRANSMITTERS


OUTPUT TRANSFORMERS
FIBRE CHANNEL 0 OUTPUT
FIBRE CHANNEL 1 OUTPUT
FIBRE CHANNEL 2 OUTPUT
FIBRE CHANNEL 3 OUTPUT
D 79986
X

APPENDIX B

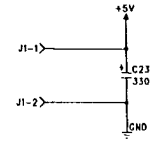
RECEIVER SCHEMATIC

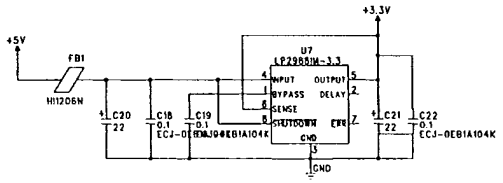
The following pages are the schematic diagram for the ADI transmitter designed for the demonstration system.

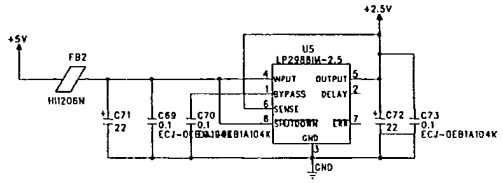
POWER CONVERSION

NOTES:

1. INTERPRET DRAWING PER MIL-STD-100.
2. THIS SCHEMATIC IS USED IN CONJUNCTION WITH CIRCUIT CARD ASSEMBLY.
3. ALL RESISTORS ARE IN OHMS, 1/8 W, 1% UNLESS OTHERWISE NOTED.
4. ALL CAPACITORS ARE IN MICROFARADS UNLESS OTHERWISE NOTED.
5. LAST COMPONENT USED:







REVISIONS

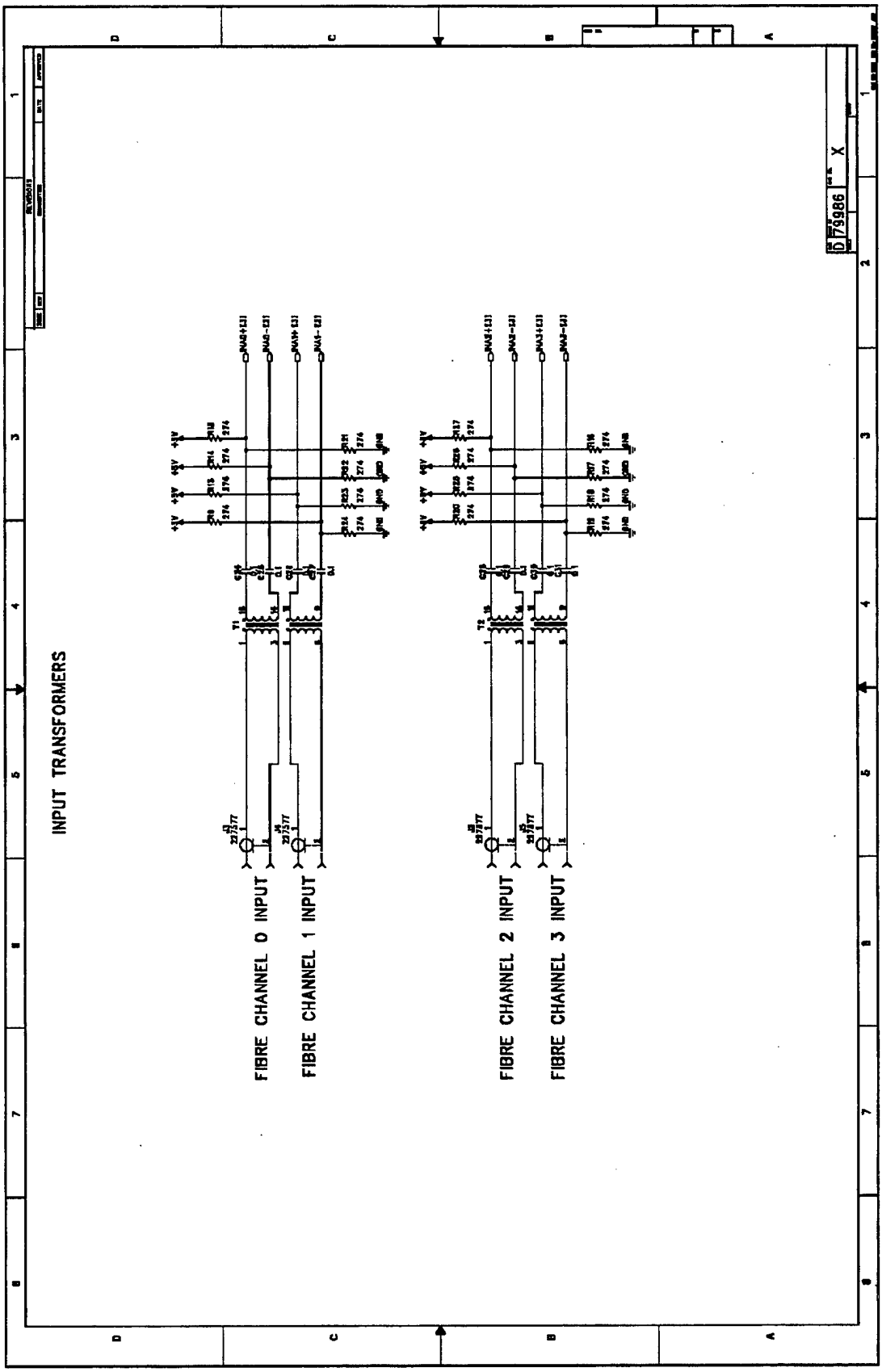
ZONE	REV	DESCRIPTION	DATE	APPROVED

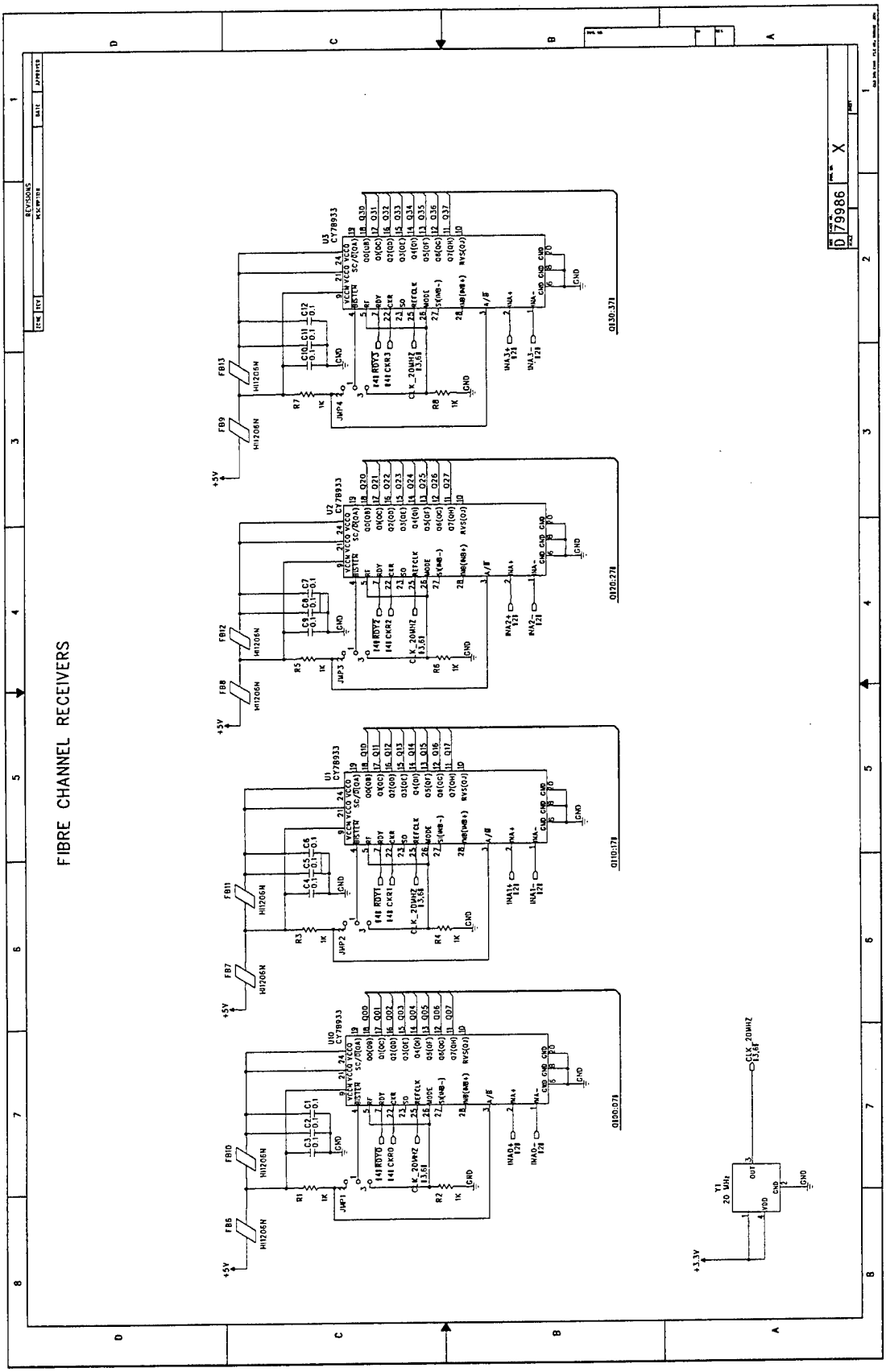
PARTS LIST

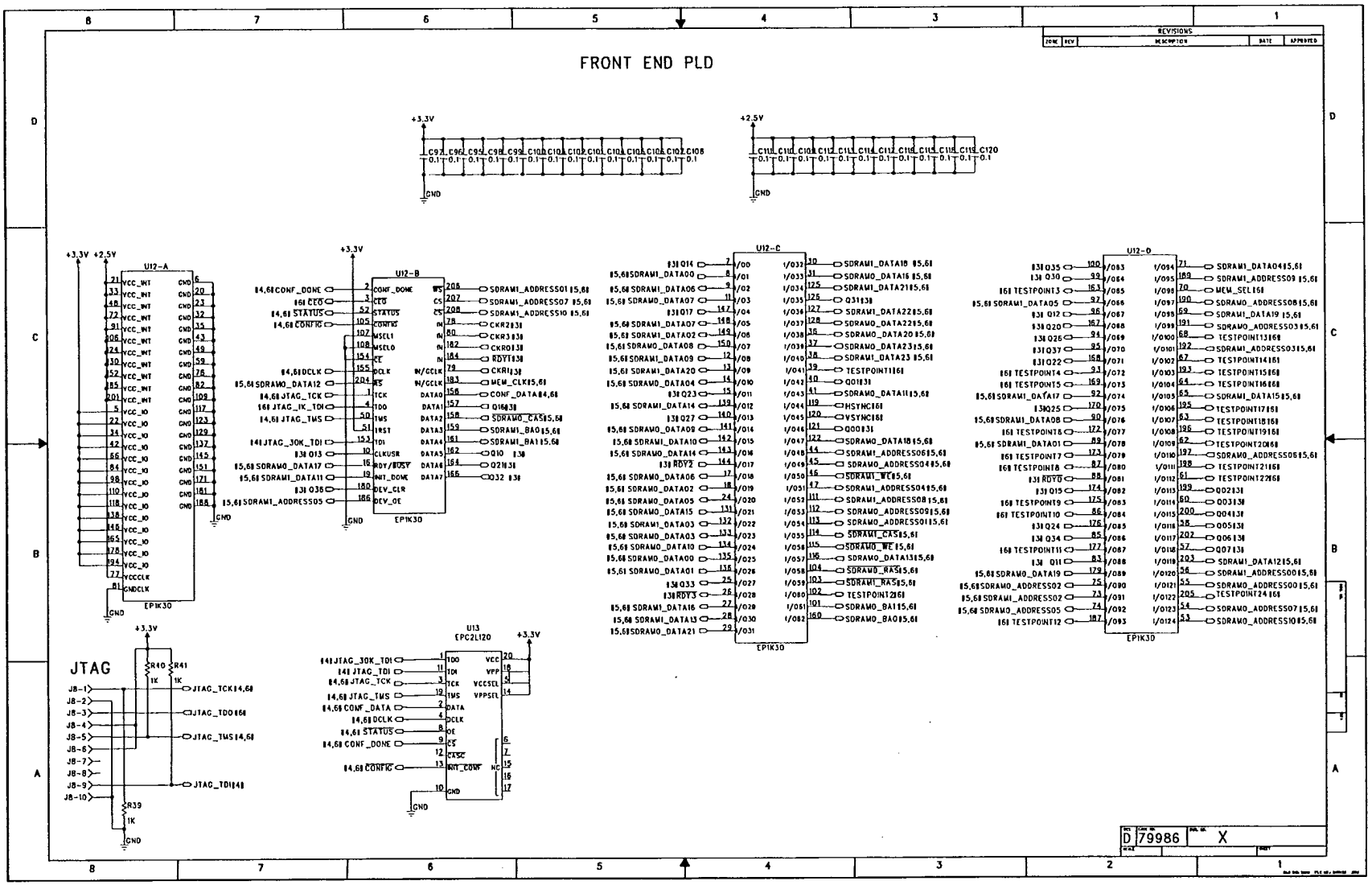
ADI
RX_MODULE

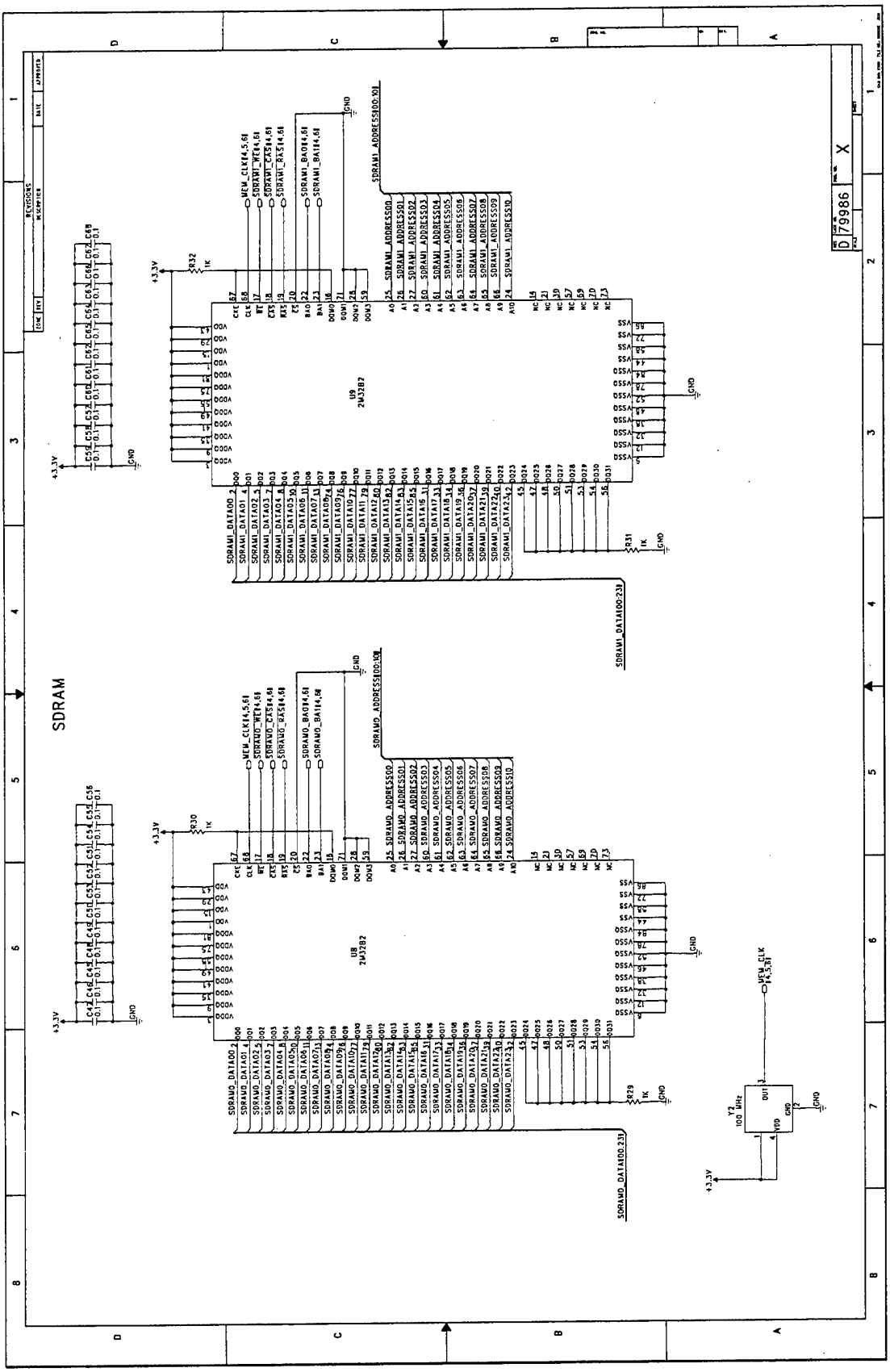
D 79986 X

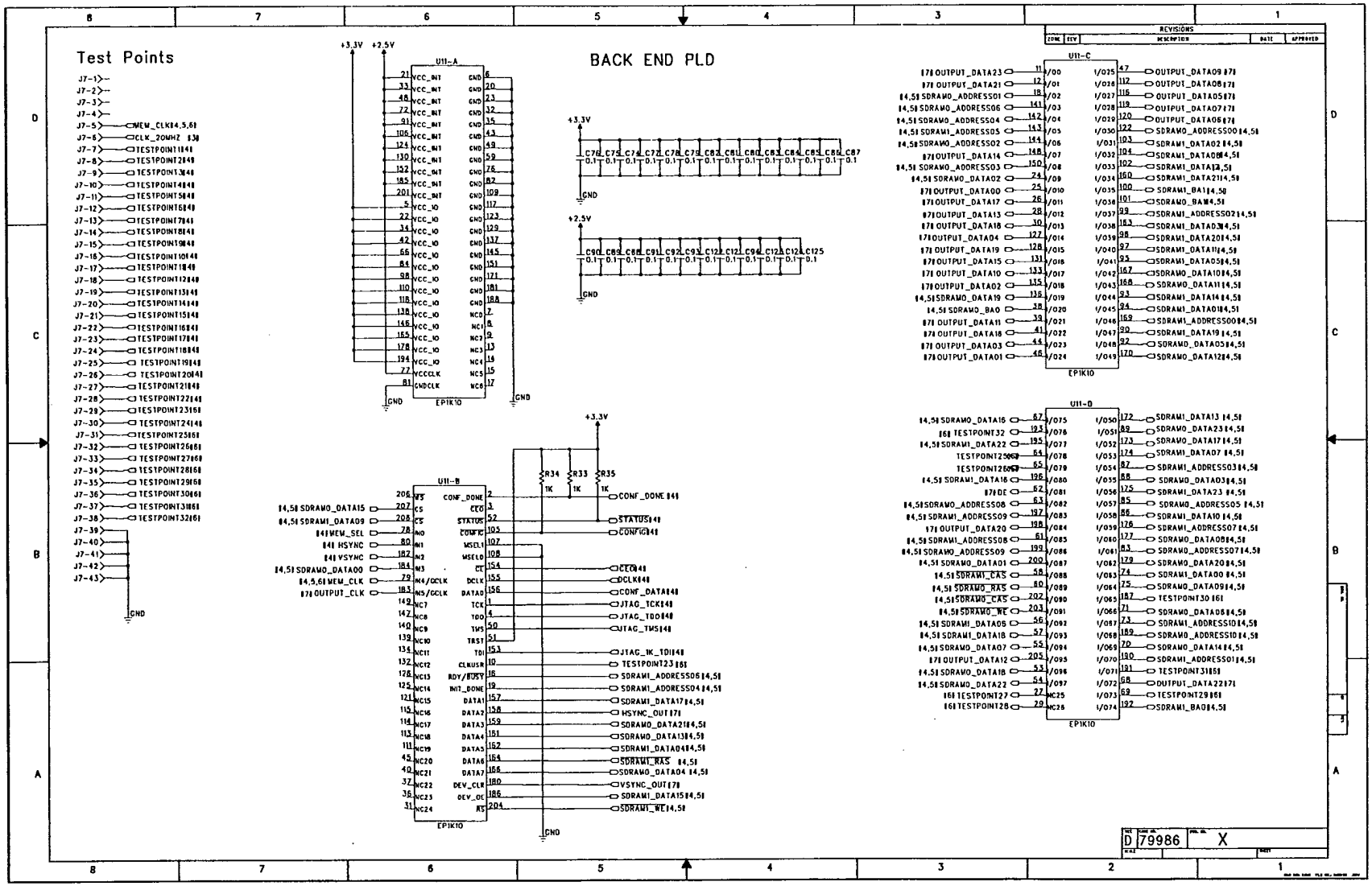
SHEET 1 OF

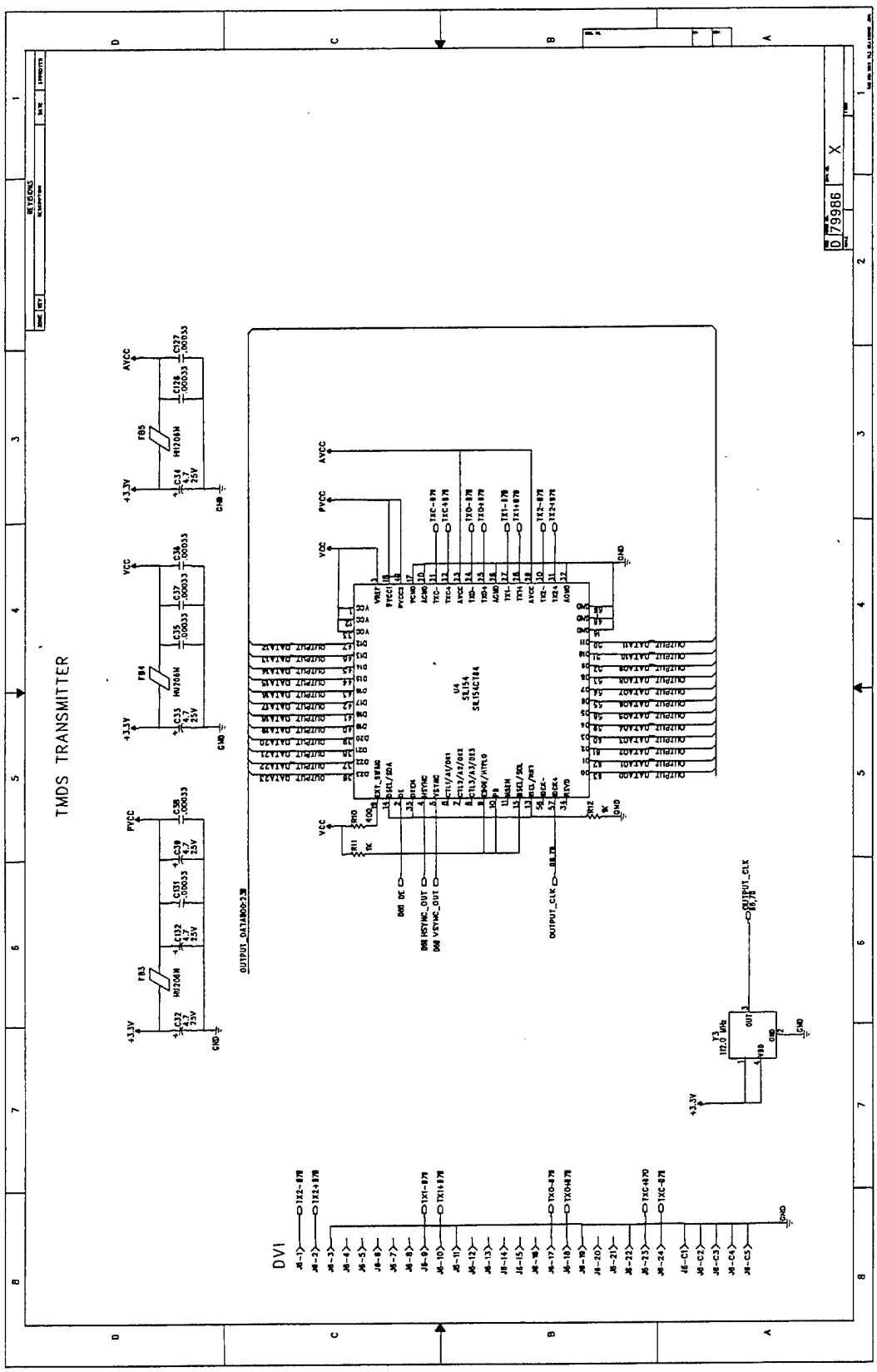

INPUT TRANSFORMERS
FIBRE CHANNEL 0 INPUT
FIBRE CHANNEL 1 INPUT
FIBRE CHANNEL 2 INPUT
FIBRE CHANNEL 3 INPUT


FIBRE CHANNEL RECEIVERS


FRONT END PLD
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
+3.3V
+2.5V
GND
U12-A
U12-B
U12-C
U12-D
EPIK30
U13
EPC2LI20
JTAG
JTAG_TCK
JTAG_TDO
JTAG_TMS
JTAG_TDI
R39
1K
D
79986
X

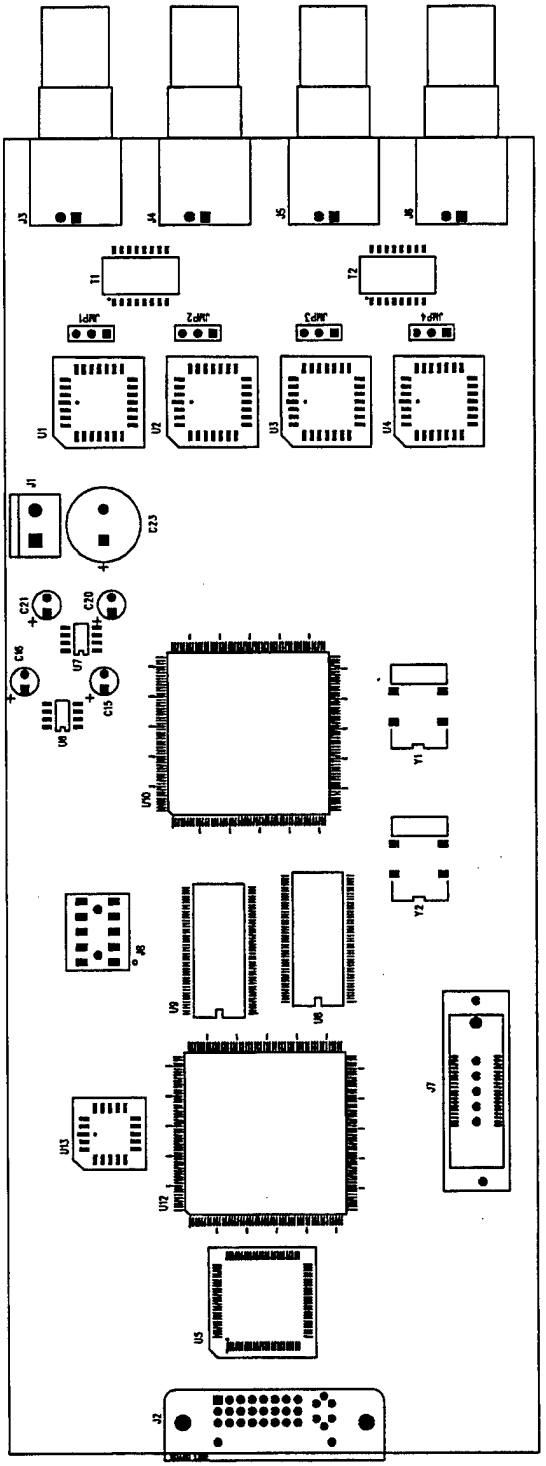

SDRAM


Test Points
BACK END PLD
U11-A
U11-B
U11-C
U11-D
EP1K10
REVISIONS
D 79986 X

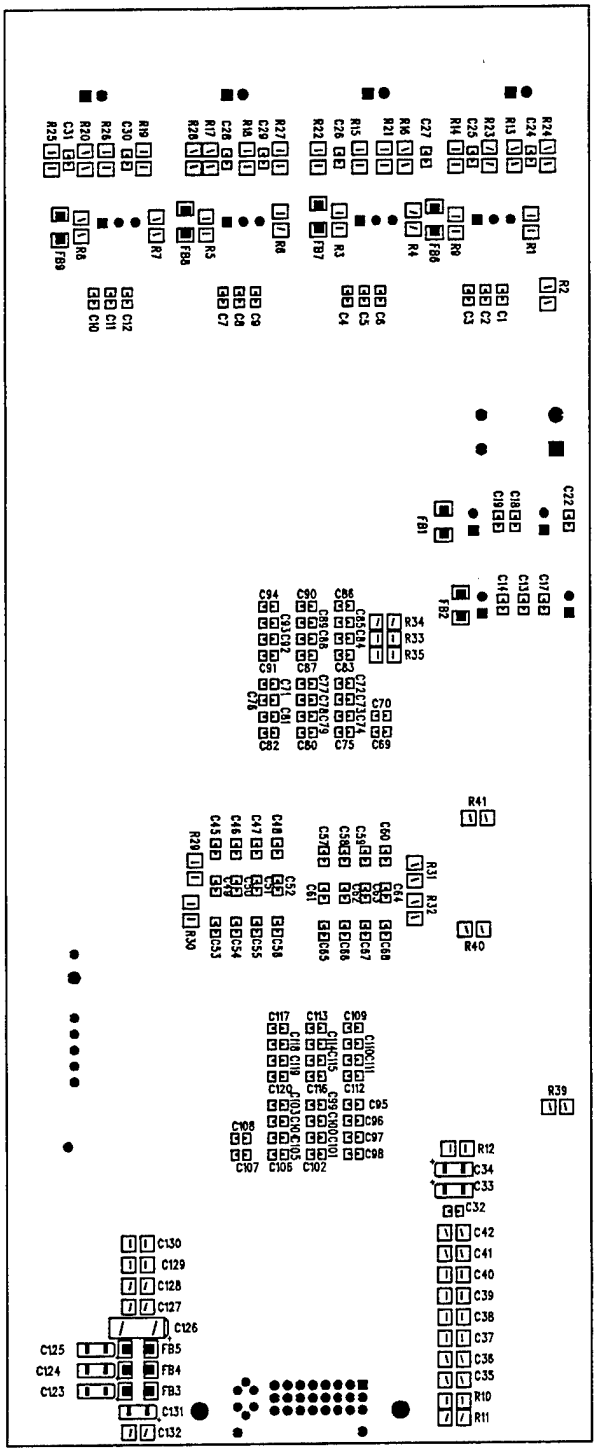

TMDS TRANSMITTER
DVI
U4
SIL154
SIL154CT84
79986
X

APPENDIX C

TRANSMITTER LAYOUT

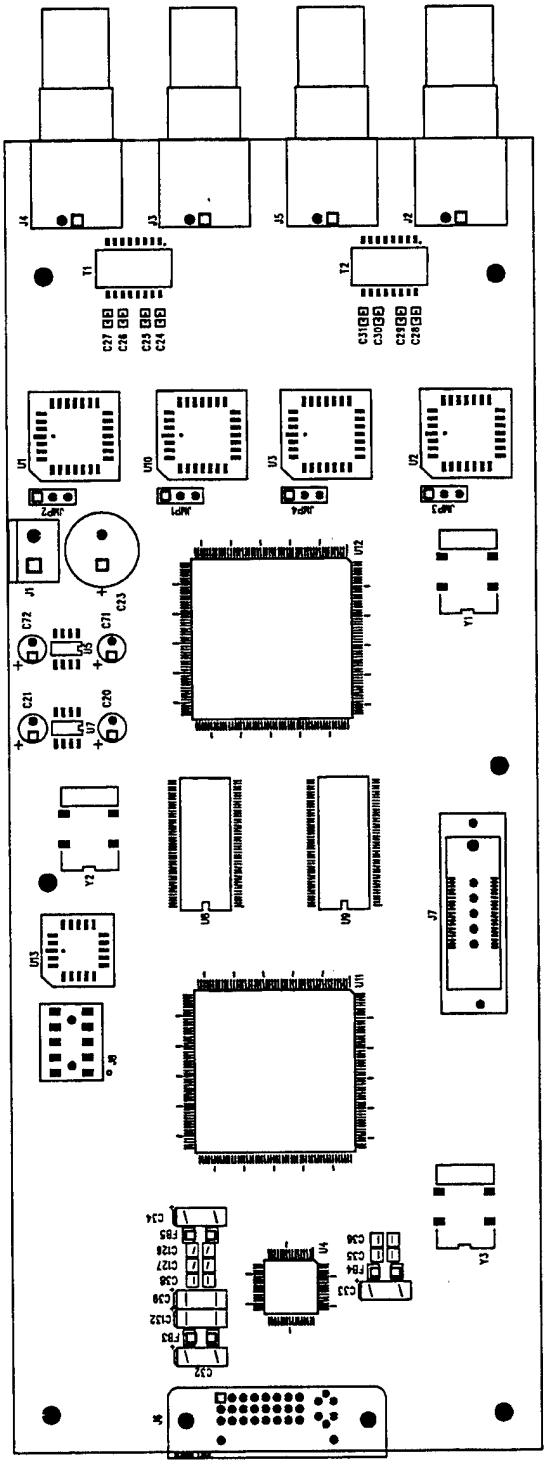


Board Dimensions:
3.5 x 8.63 in.

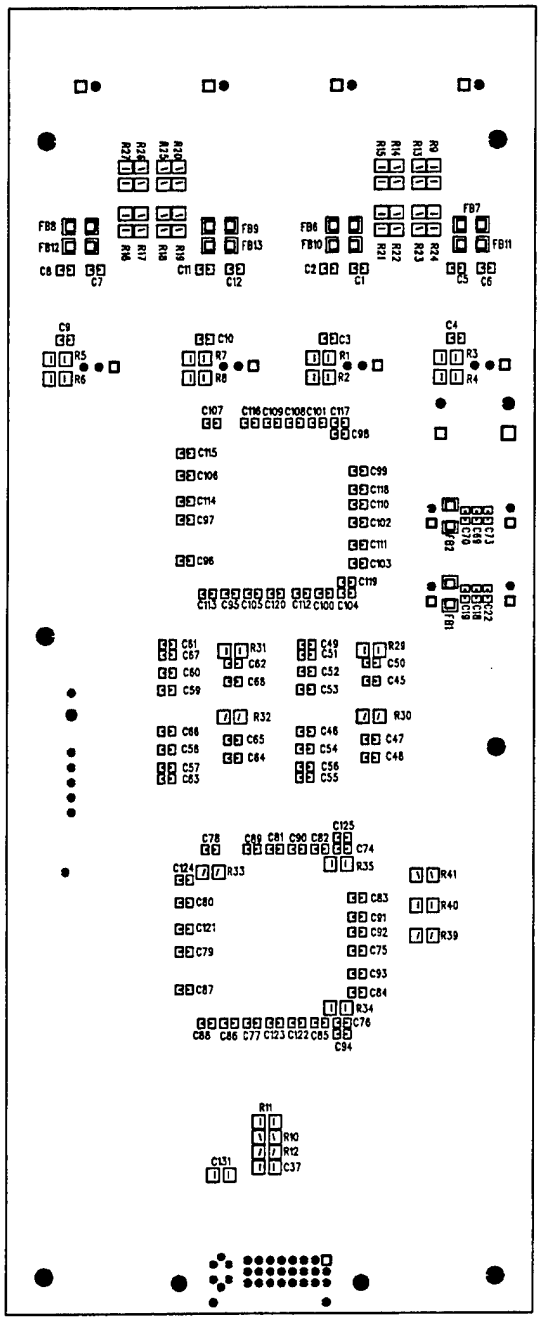


APPENDIX D

RECEIVER LAYOUT



Board Dimensions:
3.5 x 8.63 in.

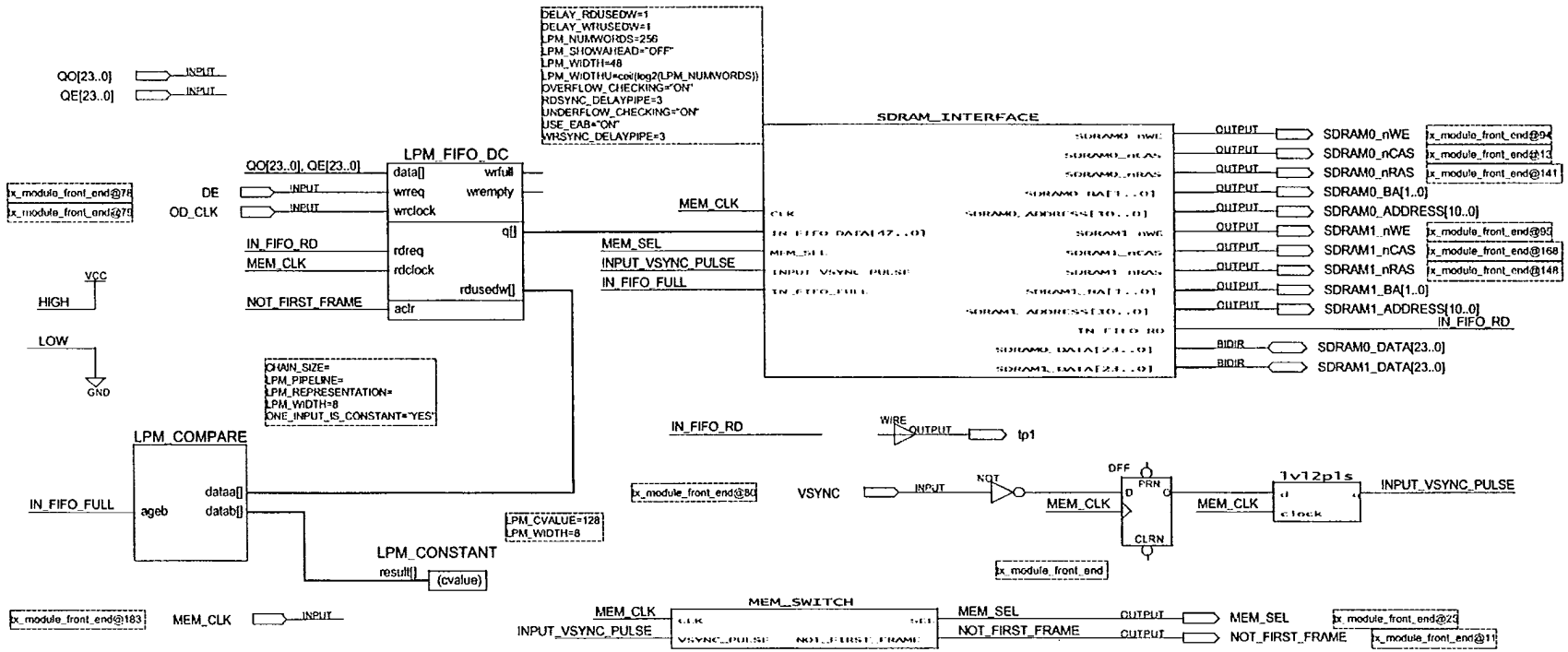


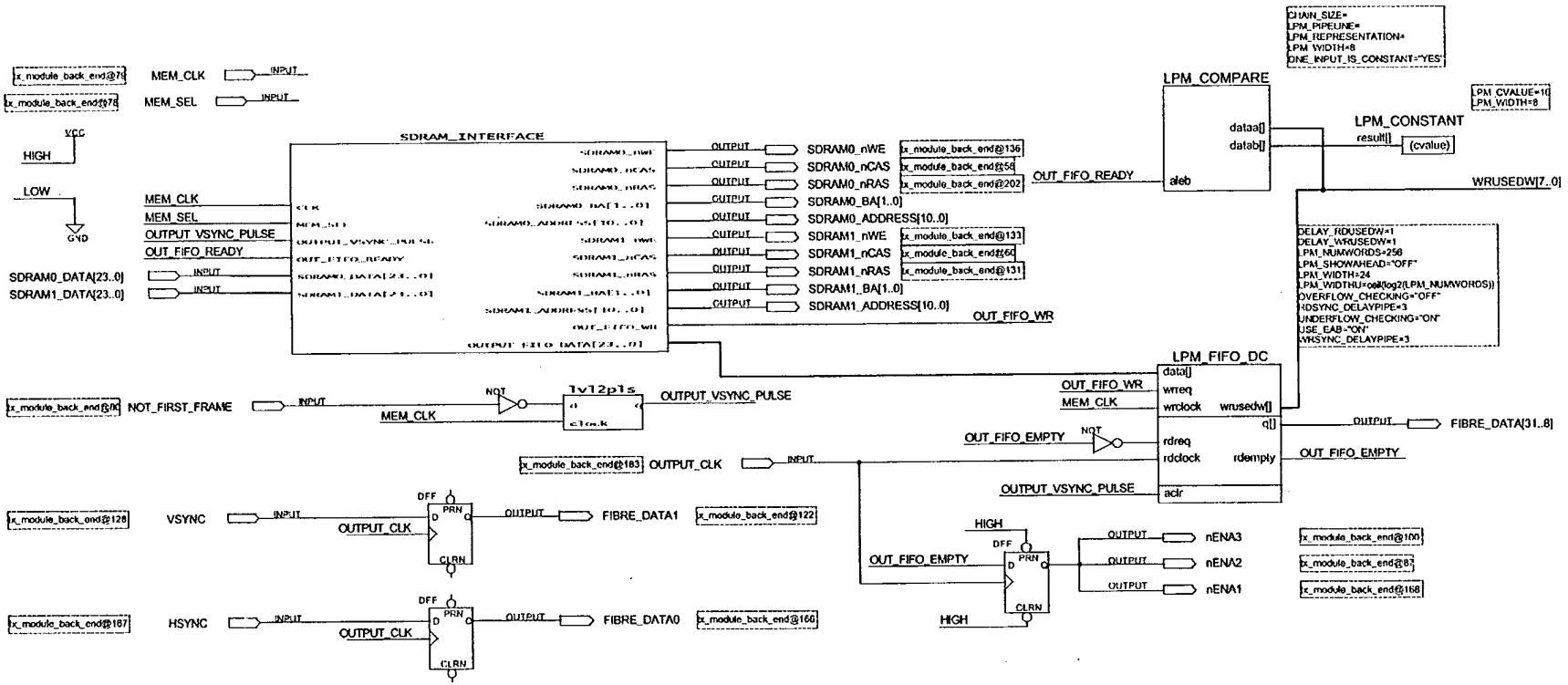
APPENDIX E

TRANSMITTER VERY HIGH-LEVEL DESCRIPTION LANGUAGE (VHDL)

The following pages are the schematic diagram for the ADI transmitter designed for the demonstration system.

The following are the VHDL source files for the ADI transmitter programmable logic devices (PLDs). The top level diagrams for the front and back end interfaces are followed by the VHDL source for each major PLD block.


QO[23..0]
QE[23..0]
INPUT
DELAY_RDUSEDW=1
DELAY_WRUSEDW=1
LPM_NUMWORDS=256
LPM_SHOWAHEAD="OFF"
LPM_WIDTH=48
LPM_WIDTHU=ceil(log2(LPM_NUMWORDS))
OVERFLOW_CHECKING="ON"
RDSYNC_DELAYPIPE=3
UNDERFLOW_CHECKING="ON"
USE_EAB="ON"
WRSYNC_DELAYPIPE=3
LPM_FIFO_DC
QO[23..0], QE[23..0]
data[]
wrfull
wrempty
wrreq
wrclock
q[]
rdreq
rdclock
rdusedw[]
aclr
DE
OD_CLK
IN_FIFO_RD
MEM_CLK
NOT_FIRST_FRAME
VCC
HIGH
LOW
GND
SDRAM_INTERFACE
MEM_CLK
MEM_SEL
INPUT_VSYNC_PULSE
IN_FIFO_FULL
OUTPUT
SDRAM0_nWE
SDRAM0_nCAS
SDRAM0_nRAS
SDRAM0_BA[1..0]
SDRAM0_ADDRESS[10..0]
SDRAM1_nWE
SDRAM1_nCAS
SDRAM1_nRAS
SDRAM1_BA[1..0]
SDRAM1_ADDRESS[10..0]
BIDIR
SDRAM0_DATA[23..0]
SDRAM1_DATA[23..0]
CHAIN_SIZE=
LPM_PIPELINE=
LPM_REPRESENTATION=
LPM_WIDTH=8
ONE_INPUT_IS_CONSTANT="YES"
LPM_COMPARE
dataa[]
datab[]
ageb
IN_FIFO_FULL
LPM_CVALUE=128
LPM_WIDTH=8
LPM_CONSTANT
result[]
(cvalue)
WIRE
tp1
VSYNC
NOT
DFF
PRN
CLRN
1v12p1s
INPUT_VSYNC_PULSE
MEM_SWITCH
NOT_FIRST_FRAME


MEM_CLK
MEM_SEL
VCC
HIGH
LOW
GND
SDRAM_INTERFACE
OUTPUT_VSYNC_PULSE
OUT_FIFO_READY
SDRAM0_DATA[23..0]
SDRAM1_DATA[23..0]
SDRAM0_nWE
SDRAM0_nCAS
SDRAM0_nRAS
SDRAM0_BA[1..0]
SDRAM0_ADDRESS[10..0]
SDRAM1_nWE
SDRAM1_nCAS
SDRAM1_nRAS
SDRAM1_BA[1..0]
SDRAM1_ADDRESS[10..0]
OUT_FIFO_WR
LPM_COMPARE
dataa[]
datab[]
aleb
LPM_CONSTANT
result[]
(cvalue)
WRUSEDW[7..0]
LPM_FIFO_DC
data[]
wrreq
wrclock
wrusedw[]
q[]
rdreq
rdclock
rdempty
aclr
OUT_FIFO_EMPTY
FIBRE_DATA[31..8]
NOT_FIRST_FRAME
OUTPUT_VSYNC_PULSE
OUTPUT_CLK
VSYNC
HSYNC
FIBRE_DATA1
FIBRE_DATA0
nENA3
nENA2
nENA1
DFF
NOT
INPUT
OUTPUT

SDRAM Input Memory Interface

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY SDRAM_INTERFACE IS     PORT
        (
                CLK : IN        STD_LOGIC;
                IN_FIFO_DATA: IN        STD_LOGIC_VECTOR(47 DOWNTO 0);
                MEM_SEL : IN STD_LOGIC;

                INPUT_VSYNC_PULSE : IN STD_LOGIC;
                IN_FIFO_FULL : IN STD_LOGIC;


                SDRAM0_nWE, SDRAM0_nCAS, SDRAM0_nRAS : OUT STD_LOGIC;
                SDRAM0_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM0_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                SDRAM1_nWE, SDRAM1_nCAS, SDRAM1_nRAS : OUT STD_LOGIC;
                SDRAM1_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM1_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                IN_FIFO_RD : OUT STD_LOGIC;

                SDRAM0_DATA, SDRAM1_DATA : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)

        );
END SDRAM_INTERFACE;

ARCHITECTURE ONE OF SDRAM_INTERFACE IS

        SIGNAL  in_fifo_rd_d : STD_LOGIC;
        SIGNAL  input_word : STD_LOGIC_VECTOR(23 DOWNTO 0);
        SIGNAL  IN_WORD_SEL : STD_LOGIC_VECTOR(0 DOWNTO 0);
        SIGNAL  nMEM_SEL: STD_LOGIC;
        SIGNAL  sdram0_mem_signals, sdram1_mem_signals : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL  input_mem_signals, registered_input_mem_signals : STD_LOGIC_VECTOR(16 DOWNTO 0);
        SIGNAL  INPUT_nWE, INPUT_nCAS, INPUT_nRAS : STD_LOGIC;
        SIGNAL  INPUT_BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL  INPUT_ADDRESS : STD_LOGIC_VECTOR(10 DOWNTO 0);
        SIGNAL   SDRAM_DATA : STD_LOGIC_2D(1 DOWNTO 0, 23 DOWNTO 0);

        COMPONENT INPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                IN_FIFO_FULL : IN STD_LOGIC;

                IN_FIFO_RD : OUT STD_LOGIC;
                IN_WORD_SEL : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
        );
        END COMPONENT;

BEGIN

        INPUT_CONTROL : INPUT_MEM_CONTROL
                PORT MAP(
                        CLK => CLK,
                        VSYNC_PULSE => INPUT_VSYNC_PULSE,
                        IN_FIFO_FULL => IN_FIFO_FULL,
```

```
                        IN_FIFO_RD => in_fifo_rd_d,
                        IN_WORD_SEL => IN_WORD_SEL(0),
                        nWE => INPUT_nWE,
                        nCAS => INPUT_nCAS,
                        nRAS => INPUT_nRAS,
                        BA => INPUT_BA,
                        ADDRESS => INPUT_ADDRESS
                );


        WORD_MUX: lpm_mux
                GENERIC MAP(LPM_WIDTH => 24, LPM_SIZE => 2, LPM_WIDTHS => 1, LPM_PIPELINE => 1)
                PORT MAP(data => SDRAM_DATA, clock => CLK, sel => IN_WORD_SEL, result =>
input_word);

        nMEM_SEL <= NOT MEM_SEL;

        DATA_BUSSES:
        FOR i IN 0 to 23 GENERATE
                SDRAM_DATA(0, i) <= IN_FIFO_DATA(i);
                SDRAM_DATA(1, i) <= IN_FIFO_DATA(24 + i);

                SDRAM0_DATA_BUS: TRI
                PORT MAP(a_in => input_word(i), oe => nMEM_SEL, a_out => SDRAM0_DATA(i));

                SDRAM1_DATA_BUS: TRI
                PORT MAP(a_in => input_word(i), oe => MEM_SEL, a_out => SDRAM1_DATA(i));
        END GENERATE;

        input_mem_signals <=
                in_fifo_rd_d &
                INPUT_nWE &
                INPUT_nCAS &
                INPUT_nRAS &
                INPUT_BA(1 DOWNTO 0) &
                INPUT_ADDRESS(10 DOWNTO 0);

        SDRAM0_nWE <= sdram0_mem_signals(15);
        SDRAM0_nCAS <= sdram0_mem_signals(14);
        SDRAM0_nRAS <= sdram0_mem_signals(13);
        SDRAM0_BA(1 DOWNTO 0) <= sdram0_mem_signals(12 DOWNTO 11);
        SDRAM0_ADDRESS(10 DOWNTO 0) <= sdram0_mem_signals(10 DOWNTO 0);

        SDRAM1_nWE <= sdram1_mem_signals(15);
        SDRAM1_nCAS <= sdram1_mem_signals(14);
        SDRAM1_nRAS <= sdram1_mem_signals(13);
        SDRAM1_BA(1 DOWNTO 0) <= sdram1_mem_signals(12 DOWNTO 11);
        SDRAM1_ADDRESS(10 DOWNTO 0) <= sdram1_mem_signals(10 DOWNTO 0);

        INPUT_REGISTER : LPM_FF
                GENERIC MAP(LPM_WIDTH => 17)
                PORT MAP(data => input_mem_signals, clock => CLK, q =>
registered_input_mem_signals);

        IN_FIFO_RD <= registered_input_mem_signals(16);

        CONTROL_BUSSES:
        FOR i IN 0 to 15 GENERATE
                SDRAM0_CONTROL_BUS: TRI
                PORT MAP(a_in => registered_input_mem_signals(i), oe => nMEM_SEL, a_out =>
sdram0_mem_signals(i));

                SDRAM1_CONTROL_BUS: TRI
                PORT MAP(a_in => registered_input_mem_signals(i), oe => MEM_SEL, a_out =>
sdram1_mem_signals(i));
        END GENERATE;


END ONE;
```

ADI Memory Switch

```
-- ADI Memory Switch

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY MEM_SWITCH IS
        PORT
        (
--              tp_hs_en, tp_vs_en: OUT STD_LOGIC;

                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;

                SEL: OUT STD_LOGIC;
                NOT_FIRST_FRAME: OUT STD_LOGIC
        );
END MEM_SWITCH;

ARCHITECTURE ONE OF MEM_SWITCH IS

        SIGNAL output_vsync: STD_LOGIC;
        SIGNAL output_vsync_pulse : STD_LOGIC;

        SIGNAL sel_ff_data, sel_ff_q : STD_LOGIC_VECTOR(0 DOWNTO 0);

        TYPE state_type IS (remaining_vsyncs, first_vsync);
        SIGNAL ps, ns : state_type := remaining_vsyncs;

        COMPONENT lvl2pls
                PORT(
                        d               : IN    STD_LOGIC;
                        clock   : IN    STD_LOGIC;
                        q               : OUT   STD_LOGIC);
        END COMPONENT;

BEGIN

        VSYNC_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => VSYNC_PULSE, cout => output_vsync);

        VSYNC_LVL2PLS: lvl2pls
                PORT MAP(d => output_vsync, clock => CLK, q => output_vsync_pulse);

        MEM_SEL_FF: lpm_ff
                GENERIC MAP(LPM_WIDTH => 1)
                PORT MAP(clock => CLK, enable => output_vsync_pulse, data => sel_ff_data, q =>
sel_ff_q);

        sel_ff_data(0) <= NOT sel_ff_q(0);
        SEL <= sel_ff_q(0);

        PROCESS(ps, output_vsync_pulse, VSYNC_PULSE)
                BEGIN
                        CASE ps IS

                                WHEN remaining_vsyncs =>
                                        NOT_FIRST_FRAME <= '1';

                                        IF output_vsync_pulse = '1' THEN
                                                ns <= first_vsync;
                                        ELSE
                                                ns <= remaining_vsyncs;
                                        END IF;

                                WHEN first_vsync =>
```

```
                                        NOT_FIRST_FRAME <= '0';

                                        IF VSYNC_PULSE = '1' THEN
                                                ns <= remaining_vsyncs;
                                        ELSE
                                                ns <= first_vsync;
                                        END IF;

                        END CASE;
        END PROCESS;

        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
                IF CLK = '1' THEN
                        ps <= ns;
                END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

SDRAM Output Memory Interface

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY SDRAM_INTERFACE IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                MEM_SEL : IN STD_LOGIC;

                OUTPUT_VSYNC_PULSE : IN STD_LOGIC;
                OUT_FIFO_READY : IN STD_LOGIC;

                SDRAM0_DATA, SDRAM1_DATA : IN STD_LOGIC_VECTOR(23 DOWNTO 0);

                SDRAM0_nWE, SDRAM0_nCAS, SDRAM0_nRAS : OUT STD_LOGIC;
                SDRAM0_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM0_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                SDRAM1_nWE, SDRAM1_nCAS, SDRAM1_nRAS : OUT STD_LOGIC;
                SDRAM1_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM1_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                OUT_FIFO_WR : OUT STD_LOGIC;
                OUTPUT_FIFO_DATA : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)

        );
END SDRAM_INTERFACE;

ARCHITECTURE ONE OF SDRAM_INTERFACE IS

        SIGNAL  nMEM_SEL : STD_LOGIC;
        SIGNAL  OUT_FIFO_WR_PIPE_IN : STD_LOGIC;
        SIGNAL  MEM_SEL_VECTOR : STD_LOGIC_VECTOR(0 DOWNTO 0);
        SIGNAL  output_mem_signals, registered_output_mem_signals, sdram0_mem_signals,
sdram1_mem_signals : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL  OUTPUT_nWE, OUTPUT_nCAS, OUTPUT_nRAS : STD_LOGIC;
        SIGNAL  OUTPUT_BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL  OUTPUT_ADDRESS : STD_LOGIC_VECTOR(10 DOWNTO 0);
        SIGNAL   SDRAM_DATA : STD_LOGIC_2D(1 DOWNTO 0, 23 DOWNTO 0);
```

```
	COMPONENT OUTPUT_MEM_CONTROL IS
	PORT
	(
		CLK : IN		STD_LOGIC;
		VSYNC_PULSE : IN		STD_LOGIC;
		OUT_FIFO_READY : IN STD_LOGIC;

		OUT_FIFO_WR : OUT STD_LOGIC;
		nWE, nCAS, nRAS : OUT STD_LOGIC;
		BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
		ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)

	);
	END COMPONENT;

BEGIN

	OUPUT_CONTROL : OUTPUT_MEM_CONTROL
		PORT MAP(
			CLK => CLK,
			VSYNC_PULSE => OUTPUT_VSYNC_PULSE,
			OUT_FIFO_READY => OUT_FIFO_READY,
			OUT_FIFO_WR => OUT_FIFO_WR_PIPE_IN,
			nWE => OUTPUT_nWE,
			nCAS => OUTPUT_nCAS,
			nRAS => OUTPUT_nRAS,
			BA => OUTPUT_BA,
			ADDRESS => OUTPUT_ADDRESS
		);

	nMEM_SEL <= NOT MEM_SEL;

	OUT_FIFO_WR_PIPE : LPM_SHIFTREG
		GENERIC MAP(LPM_WIDTH => 4)
		PORT MAP(shiftin => OUT_FIFO_WR_PIPE_IN, clock => CLK, shiftout => OUT_FIFO_WR);

	output_mem_signals <=
		OUTPUT_nWE &
		OUTPUT_nCAS &
		OUTPUT_nRAS &
		OUTPUT_BA(1 DOWNTO 0) &
		OUTPUT_ADDRESS(10 DOWNTO 0);

	INPUT_REGISTER : LPM_FF
		GENERIC MAP(LPM_WIDTH => 16)
		PORT MAP(data => output_mem_signals, clock => CLK, q =>
registered_output_mem_signals);

	CONTROL_BUSSES:
	FOR i IN 0 TO 15 GENERATE
		SDRAM0_CONTROL_BUS: TRI
		PORT MAP(a_in => registered_output_mem_signals(i), oe => nMEM_SEL, a_out =>
sdram0_mem_signals(i));

		SDRAM1_CONTROL_BUS: TRI
		PORT MAP(a_in => registered_output_mem_signals(i), oe => MEM_SEL, a_out =>
sdram1_mem_signals(i));
	END GENERATE;

	SDRAM0_nWE <= sdram0_mem_signals(15);
	SDRAM0_nCAS <= sdram0_mem_signals(14);
	SDRAM0_nRAS <= sdram0_mem_signals(13);
	SDRAM0_BA(1 DOWNTO 0) <= sdram0_mem_signals(12 DOWNTO 11);
	SDRAM0_ADDRESS(10 DOWNTO 0) <= sdram0_mem_signals(10 DOWNTO 0);

	SDRAM1_nWE <= sdram1_mem_signals(15);
	SDRAM1_nCAS <= sdram1_mem_signals(14);
	SDRAM1_nRAS <= sdram1_mem_signals(13);
	SDRAM1_BA(1 DOWNTO 0) <= sdram1_mem_signals(12 DOWNTO 11);
	SDRAM1_ADDRESS(10 DOWNTO 0) <= sdram1_mem_signals(10 DOWNTO 0);
```

```
        DATA_BUSSES:
        FOR i IN 0 to 23 GENERATE
                SDRAM_DATA(0, i) <= SDRAM0_DATA(i);
                SDRAM_DATA(1, i) <= SDRAM1_DATA(i);
        END GENERATE;

        MEM_SEL_VECTOR(0) <= MEM_SEL;

        OUTPUT_FIFO_DATA_MUX : lpm_mux
                GENERIC MAP(LPM_WIDTH => 24, LPM_SIZE => 2, LPM_WIDTHS => 1, LPM_PIPELINE => 1)
                PORT MAP(data => SDRAM_DATA, clock => CLK, sel => MEM_SEL_VECTOR, result =>
OUTPUT_FIFO_DATA);

END ONE;
```

ADI Input Memory Controller

```
-- ADI Input Memory Controller
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY INPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                IN_FIFO_FULL : IN STD_LOGIC;

                IN_FIFO_RD : OUT STD_LOGIC;
                IN_WORD_SEL : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)

        );
END INPUT_MEM_CONTROL;

ARCHITECTURE ONE OF INPUT_MEM_CONTROL IS
        CONSTANT NUM_REFRESH_CYCLES: STD_LOGIC_VECTOR := "111";

        SIGNAL refresh_en, refresh_load, refresh_rdy : STD_LOGIC;
        SIGNAL refresh_cnt : STD_LOGIC_VECTOR(2 downto 0);

        TYPE init_state_type IS (start, wait_for_frame, precharge, refresh_nop, auto_refresh1,
wait_for_refresh1, auto_refresh2, wait_for_refresh2, load_mode_reg, done);
        SIGNAL init_ps, init_ns : init_state_type := start;

        TYPE memory_state_type IS (initialize, normal_operation);
        SIGNAL memory_ps, memory_ns : memory_state_type := initialize;

        TYPE normal_op_state_type IS (wait_for_FIFO, active, pre_write_nop, write_first_word,
burst, last_word, burst_terminate, precharge_nop1, precharge_nop2, precharge_nop3, refresh,
refresh_nop, auto_refresh, wait_for_refresh);
        SIGNAL normal_op_ps, normal_op_ns : normal_op_state_type := wait_for_FIFO;

        TYPE sdram_cmd_type IS (nop, active, read, write, precharge, refresh, load_mode_reg,
burst_terminate);
        SIGNAL sdram_cmd : sdram_cmd_type;
        SIGNAL test : std_logic_vector(2 DOWNTO 0);

        SIGNAL row_rdy, column_rdy, column_en, row_en, bank_en : STD_LOGIC;
        SIGNAL bank_count : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL row_count, row_count_ready : STD_LOGIC_VECTOR(10 DOWNTO 0);
```

```
        SIGNAL column_count : STD_LOGIC_VECTOR(7 DOWNTO 0);

        SIGNAL word_count_en, word_count_clr, word_count_rdy : STD_LOGIC;

BEGIN

        nRAS <= test(2);
        nCAS <= test(1);
        nWE <= test(0);

        WITH sdram_cmd SELECT
                test <=
                                "111" WHEN nop,
                                "011" WHEN active,
                                "101" WHEN read,
                                "100" WHEN write,
                                "010" WHEN precharge,
                                "001" WHEN refresh,
                                "110" WHEN burst_terminate,
                                "000" WHEN load_mode_reg;

        AUTO_REFRESH_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 3, LPM_DIRECTION => "DOWN")
                PORT MAP(data => refresh_cnt, clock => CLK, cnt_en => refresh_en, sload =>
refresh_load, cout => refresh_rdy);

        COLUMN_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 8, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => column_en, q => column_count, sclr =>
VSYNC_PULSE, cout => column_rdy);

        ROW_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 11, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => row_en, q => row_count, sclr => VSYNC_PULSE);

        ROW_COMPARE : lpm_compare
                GENERIC MAP(LPM_WIDTH => 11)
                PORT MAP(dataa => row_count, datab => row_count_ready, aeb => row_rdy);

        ROW_COUNT_READY_CONSTANT : lpm_constant
                GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 1280)
                PORT MAP(result => row_count_ready);

        BANK_COUNTER: lpm_counter
                GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => bank_en, q => bank_count, sclr => VSYNC_PULSE);

        WORD_COUNTER: lpm_counter
                GENERIC MAP(LPM_WIDTH => 1, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => word_count_en, sclr => word_count_clr, cout =>
word_count_rdy);

        SM:
        PROCESS(memory_ps, init_ps, normal_op_ps, VSYNC_PULSE, refresh_rdy)
        BEGIN
                CASE memory_ps IS
                        WHEN initialize =>
                                IN_FIFO_RD <= '0';
                                IN_WORD_SEL <= '0';
                                normal_op_ns <= wait_for_FIFO;
                                column_en <= '0';
                                row_en <= '0';
                                bank_en <= '0';
                                word_count_en <= '0';
                                word_count_clr <= '0';

                                CASE init_ps IS
                                        WHEN start=>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
```

```
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        IF VSYNC_PULSE= '1' THEN
                init_ns <= wait_for_frame;
        ELSE
                init_ns <= start;
        END IF;
WHEN wait_for_frame=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        IF VSYNC_PULSE= '1' THEN
                init_ns <= precharge;
        ELSE
                init_ns <= wait_for_frame;
        END IF;

WHEN precharge=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= refresh_nop;

WHEN refresh_nop=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= auto_refresh1;

WHEN auto_refresh1=>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= wait_for_refresh1;
WHEN wait_for_refresh1=>
        refresh_cnt <= "000";
        refresh_en <= '1';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        IF refresh_rdy= '1' THEN
                init_ns <= auto_refresh2;
        ELSE
                init_ns <= wait_for_refresh1;
        END IF;
```

```
                WHEN auto_refresh2=>
                        refresh_cnt <= NUM_REFRESH_CYCLES;
                        refresh_en <= '0';
                        refresh_load <= '1';
                        sdram_cmd <= refresh;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        init_ns <= wait_for_refresh2;
                WHEN wait_for_refresh2=>
                        refresh_cnt <= "000";
                        refresh_en <= '1';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        IF refresh_rdy= '1' THEN
                                init_ns <= load_mode_reg;
                        ELSE
                                init_ns <= wait_for_refresh2;
                        END IF;

                WHEN load_mode_reg=>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= load_mode_reg;
                        ADDRESS <= "00000110111";
                        BA <= "00";

                        memory_ns <= initialize;
                        init_ns <= done;

                WHEN done=>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        init_ns <= done;
                        memory_ns <= normal_operation;
        END CASE;

WHEN normal_operation =>

        memory_ns <= normal_operation;
        init_ns <= done;

        CASE normal_op_ps IS
                WHEN wait_for_FIFO =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";
                        IN_WORD_SEL <= '0';
                        column_en <= '0';
                        row_en <= '0';
                        bank_en <= '0';
                        word_count_en <= '0';
                        word_count_clr <= '1';
                        IN_FIFO_RD <= '0';

                        IF IN_FIFO_FULL = '1' THEN
                                normal_op_ns <= active;
```

```
        ELSE
                normal_op_ns <= wait_for_FIFO;
        END IF;

WHEN active =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= active;
        ADDRESS <= row_count;
        BA <= bank_count;
        IN_FIFO_RD <= '1';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= pre_write_nop;

WHEN pre_write_nop =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= write_first_word;


WHEN write_first_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= write;
        ADDRESS <= "10000000000";
        BA <= bank_count;
        IN_FIFO_RD <= '1';
        IN_WORD_SEL <= '0';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';
        word_count_clr <= '0';

        normal_op_ns <= burst;

WHEN burst =>
        refresh_load <= '0';
        refresh_cnt <= "000";
        refresh_en <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_WORD_SEL <= word_count_rdy;
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';
        word_count_clr <= '0';

        IF column_rdy = '1' THEN
```

```
                IN_FIFO_RD <= '0';
                normal_op_ns <= last_word;
        ELSE
                IN_FIFO_RD <= NOT word_count_rdy;
                normal_op_ns <= burst;
        END IF;

WHEN last_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= word_count_rdy;
        word_count_clr <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';

        normal_op_ns <= burst_terminate;

WHEN burst_terminate =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= burst_terminate;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '1';
        bank_en <= row_rdy;
        word_count_en <= '1';
        word_count_clr <= '0';

        normal_op_ns <= precharge_nop1 ;

WHEN precharge_nop1 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= precharge_nop2;

WHEN precharge_nop2 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
```

```
        word_count_clr <= '0';

        normal_op_ns <= precharge_nop3;

WHEN precharge_nop3 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= refresh;

WHEN refresh=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= refresh_nop;

WHEN refresh_nop=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= auto_refresh;

WHEN auto_refresh =>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        IN_WORD_SEL <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= wait_for_refresh;
```

```
                                        WHEN wait_for_refresh =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '1';
                                                refresh_load <= '0';
                                                sdram_cmd <= nop;
                                                ADDRESS <= "00000000000";
                                                BA <= "00";
                                                IN_FIFO_RD <= '0';
                                                IN_WORD_SEL <= '0';
                                                column_en <= '0';
                                                row_en <= '0';
                                                bank_en <= '0';
                                                word_count_en <= '0';
                                                word_count_clr <= '0';

                                                IF refresh_rdy= '1' THEN
                                                        normal_op_ns <= wait_for_FIFO;
                                                ELSE
                                                        normal_op_ns <= wait_for_refresh;
                                                END IF;

                                END CASE;
                END CASE;
        END PROCESS SM;


        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
                IF CLK = '1' THEN
                        memory_ps <= memory_ns;
                        init_ps <= init_ns;
                        normal_op_ps <= normal_op_ns;
                END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

ADI Output Memory Controller

```
-- ADI Output Memory Controller
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY OUTPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                OUT_FIFO_READY : IN STD_LOGIC;

                OUT_FIFO_WR : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)

        );
END OUTPUT_MEM_CONTROL;

ARCHITECTURE ONE OF OUTPUT_MEM_CONTROL IS
        CONSTANT NUM_REFRESH_CYCLES: STD_LOGIC_VECTOR := "111";

        SIGNAL  refresh_en, refresh_load, refresh_rdy : STD_LOGIC;
```

```
	SIGNAL refresh_cnt : STD_LOGIC_VECTOR(2 downto 0);

	TYPE init_state_type IS (
		start,
		wait_for_frame,
		precharge,
		refresh_nop,
		auto_refresh1,
		wait_for_refresh1,
		auto_refresh2,
		wait_for_refresh2,
		load_mode_reg,
		done
	);
	SIGNAL init_ps, init_ns : init_state_type := start;

	TYPE memory_state_type IS (initialize, normal_operation);
	SIGNAL memory_ps, memory_ns : memory_state_type := initialize;

	TYPE normal_op_state_type IS (
		wait_for_vsync,
		wait_for_FIFO,
		active,
		pre_read_nop,
		read_first_word,
		burst,
		read_last_word,
		burst_terminate,
		precharge_nop1,
		precharge_nop2,
		precharge_nop3,
		refresh,
		refresh_nop,
		auto_refresh,
		wait_for_refresh
	);

	SIGNAL normal_op_ns : normal_op_state_type := wait_for_vsync;
	SIGNAL normal_op_ps : normal_op_state_type := wait_for_vsync;

	TYPE sdram_cmd_type IS (
		nop,
		active,
		read,
		write,
		precharge,
		refresh,
		load_mode_reg,
		burst_terminate
	);

	SIGNAL sdram_cmd : sdram_cmd_type;
	SIGNAL test : std_logic_vector(2 DOWNTO 0);

	SIGNAL row_rdy, column_rdy, column_en, row_en, bank_en, bank_rdy, frame_en, frame_rdy :
STD_LOGIC;
	SIGNAL bank_count : STD_LOGIC_VECTOR(1 DOWNTO 0);
	SIGNAL row_count, row_count_ready : STD_LOGIC_VECTOR(10 DOWNTO 0);
	SIGNAL column_count : STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN

	nRAS <= test(2);
	nCAS <= test(1);
	nWE <= test(0);

	WITH sdram_cmd SELECT
		test <=
				"111" WHEN nop,
				"011" WHEN active,
				"101" WHEN read,
```

```
                        "100" WHEN write,
                        "010" WHEN precharge,
                        "001" WHEN refresh,
                        "110" WHEN burst_terminate,
                        "000" WHEN load_mode_reg;


        AUTO_REFRESH_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 3, LPM_DIRECTION => "DOWN")
                PORT MAP(data => refresh_cnt, clock => CLK, cnt_en => refresh_en, sload =>
refresh_load, cout => refresh_rdy);

        COLUMN_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 8, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => column_en, q => column_count, sclr =>
VSYNC_PULSE, cout => column_rdy);

        ROW_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 11, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => row_en, q => row_count, sclr => VSYNC_PULSE);

        ROW_COMPARE : lpm_compare
                GENERIC MAP(LPM_WIDTH => 11)
                PORT MAP(dataa => row_count, datab => row_count_ready, aeb => row_rdy);

        ROW_COUNT_READY_CONSTANT : lpm_constant
                GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 1280)
                PORT MAP(result => row_count_ready);

        BANK_COUNTER: lpm_counter
                GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => bank_en, q => bank_count, sclr => VSYNC_PULSE,
cout => bank_rdy);

        FRAME_COUNTER: lpm_counter
                GENERIC MAP(LPM_WIDTH => 1, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => frame_en, sclr => VSYNC_PULSE,  cout =>
frame_rdy);

        SM:
        PROCESS(memory_ps, init_ps, normal_op_ps, VSYNC_PULSE, refresh_rdy, OUT_FIFO_READY,
column_rdy, frame_rdy)
        BEGIN
                CASE memory_ps IS
                        WHEN initialize =>
                                OUT_FIFO_WR <= '0';
                                normal_op_ns <= wait_for_vsync;
                                column_en <= '0';
                                row_en <= '0';
                                bank_en <= '0';
                                frame_en <= '0';

                                CASE init_ps IS
                                        WHEN start =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
                                                sdram_cmd <= nop;
                                                ADDRESS <= "00000000000";
                                                BA <= "00";

                                                memory_ns <= initialize;
                                                IF VSYNC_PULSE= '1' THEN
                                                        init_ns <= wait_for_frame;
                                                ELSE
                                                        init_ns <= start;
                                                END IF;

                                        WHEN wait_for_frame =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
```

```
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        IF VSYNC_PULSE= '1' THEN
                init_ns <= precharge;
        ELSE
                init_ns <= wait_for_frame;
        END IF;

WHEN precharge =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= refresh_nop;

WHEN refresh_nop =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= auto_refresh1;

WHEN auto_refresh1 =>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= wait_for_refresh1;
WHEN wait_for_refresh1 =>
        refresh_cnt <= "000";
        refresh_en <= '1';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        IF refresh_rdy= '1' THEN
                init_ns <= auto_refresh2;
        ELSE
                init_ns <= wait_for_refresh1;
        END IF;
WHEN auto_refresh2 =>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";

        memory_ns <= initialize;
        init_ns <= wait_for_refresh2;
WHEN wait_for_refresh2 =>
        refresh_cnt <= "000";
        refresh_en <= '1';
```

```
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        IF refresh_rdy= '1' THEN
                                init_ns <= load_mode_reg;
                        ELSE
                                init_ns <= wait_for_refresh2;
                        END IF;

                WHEN load_mode_reg =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= load_mode_reg;
                        ADDRESS <= "00000110111";
                        BA <= "00";

                        memory_ns <= initialize;
                        init_ns <= done;

                WHEN done =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        init_ns <= done;
                        memory_ns <= normal_operation;
        END CASE;

WHEN normal_operation =>
        memory_ns <= normal_operation;
        init_ns <= done;

        CASE normal_op_ps IS
                WHEN wait_for_FIFO =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";
                        column_en <= '0';
                        row_en <= '0';
                        bank_en <= '0';
                        frame_en <= '0';
                        OUT_FIFO_WR <= '0';

                        IF OUT_FIFO_READY = '1' THEN
                                normal_op_ns <= active;
                        ELSE
                                normal_op_ns <= wait_for_FIFO;
                        END IF;

                WHEN active =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= active;
                        ADDRESS <= row_count;
                        BA <= bank_count;
                        OUT_FIFO_WR <= '0';
                        column_en <= '0';
                        row_en <= '0';
                        bank_en <= '0';
                        frame_en <= '0';
```

```
        normal_op_ns <= pre_read_nop;

WHEN pre_read_nop =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= read_first_word;


WHEN read_first_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= read;
        ADDRESS <= "10000000000";
        BA <= bank_count;
        OUT_FIFO_WR <= '0';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= burst;

WHEN burst =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '1';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        IF column_rdy = '1' THEN
                normal_op_ns <= read_last_word;
        ELSE
                normal_op_ns <= burst;
        END IF;

WHEN read_last_word=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '1';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= burst_terminate;

WHEN burst_terminate =>
        refresh_cnt <= "000";
        refresh_en <= '0';
```

```
        refresh_load <= '0';
        sdram_cmd <= burst_terminate;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '1';
        column_en <= '0';
        row_en <= '1';
        bank_en <= row_rdy;
        frame_en <= bank_rdy AND row_rdy;

        normal_op_ns <= precharge_nop1 ;

WHEN precharge_nop1 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= precharge_nop2;

WHEN precharge_nop2 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= precharge_nop3;

WHEN precharge_nop3 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= refresh;

WHEN refresh=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= refresh_nop;
```

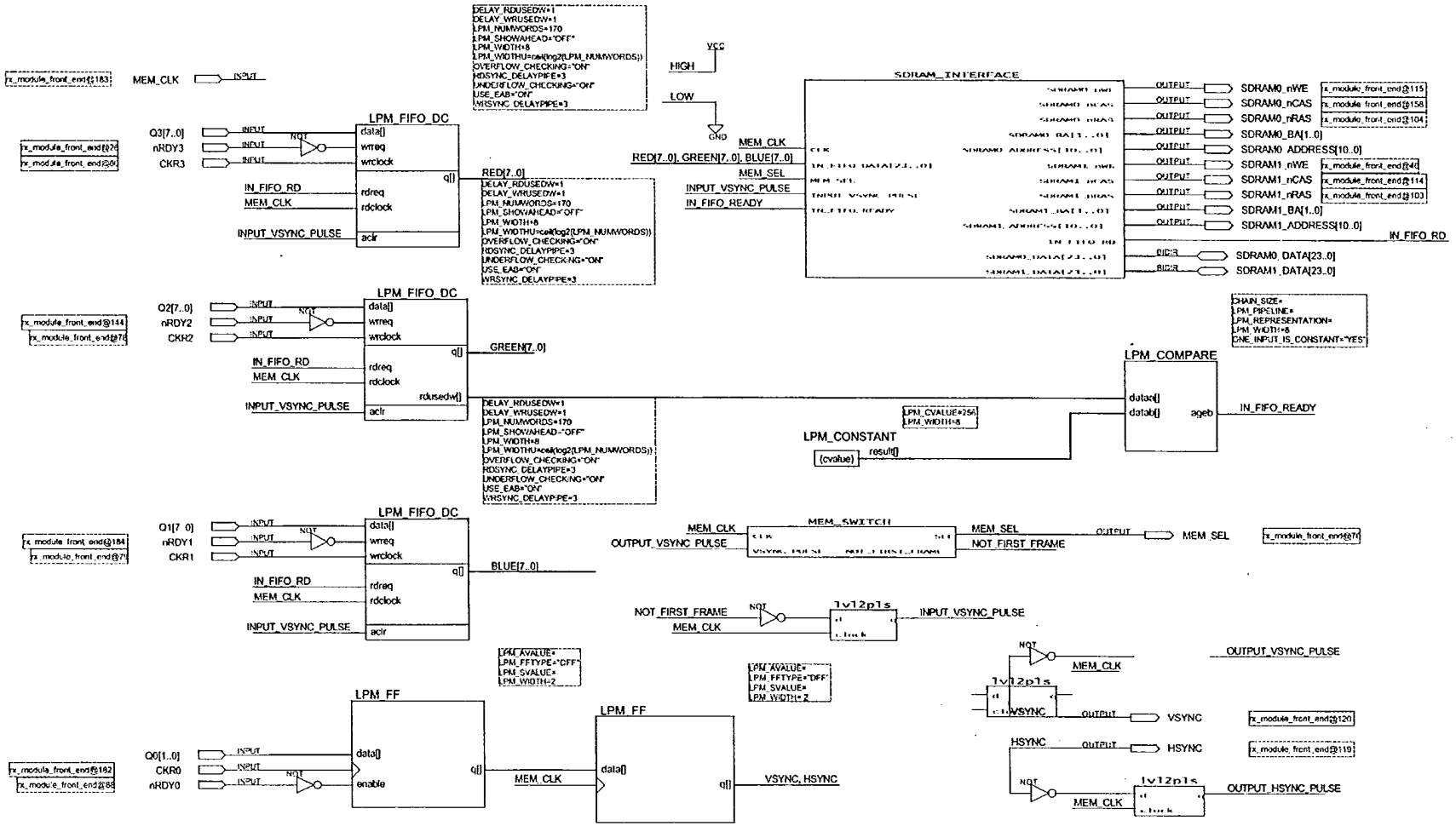
```
WHEN refresh_nop=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= auto_refresh;

WHEN auto_refresh =>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        normal_op_ns <= wait_for_refresh;

WHEN wait_for_refresh =>
        refresh_cnt <= "000";
        refresh_en <= '1';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        IF refresh_rdy= '1' THEN
                IF frame_rdy = '1' THEN
                        normal_op_ns <= wait_for_vsync;
                ELSE
                        normal_op_ns <= wait_for_FIFO;
                END IF;
        ELSE
                normal_op_ns <= wait_for_refresh;
        END IF;

WHEN wait_for_vsync =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        OUT_FIFO_WR <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        frame_en <= '0';

        IF VSYNC_PULSE = '1' THEN
                normal_op_ns <= wait_for_FIFO;
        ELSE
                normal_op_ns <= wait_for_vsync;
```

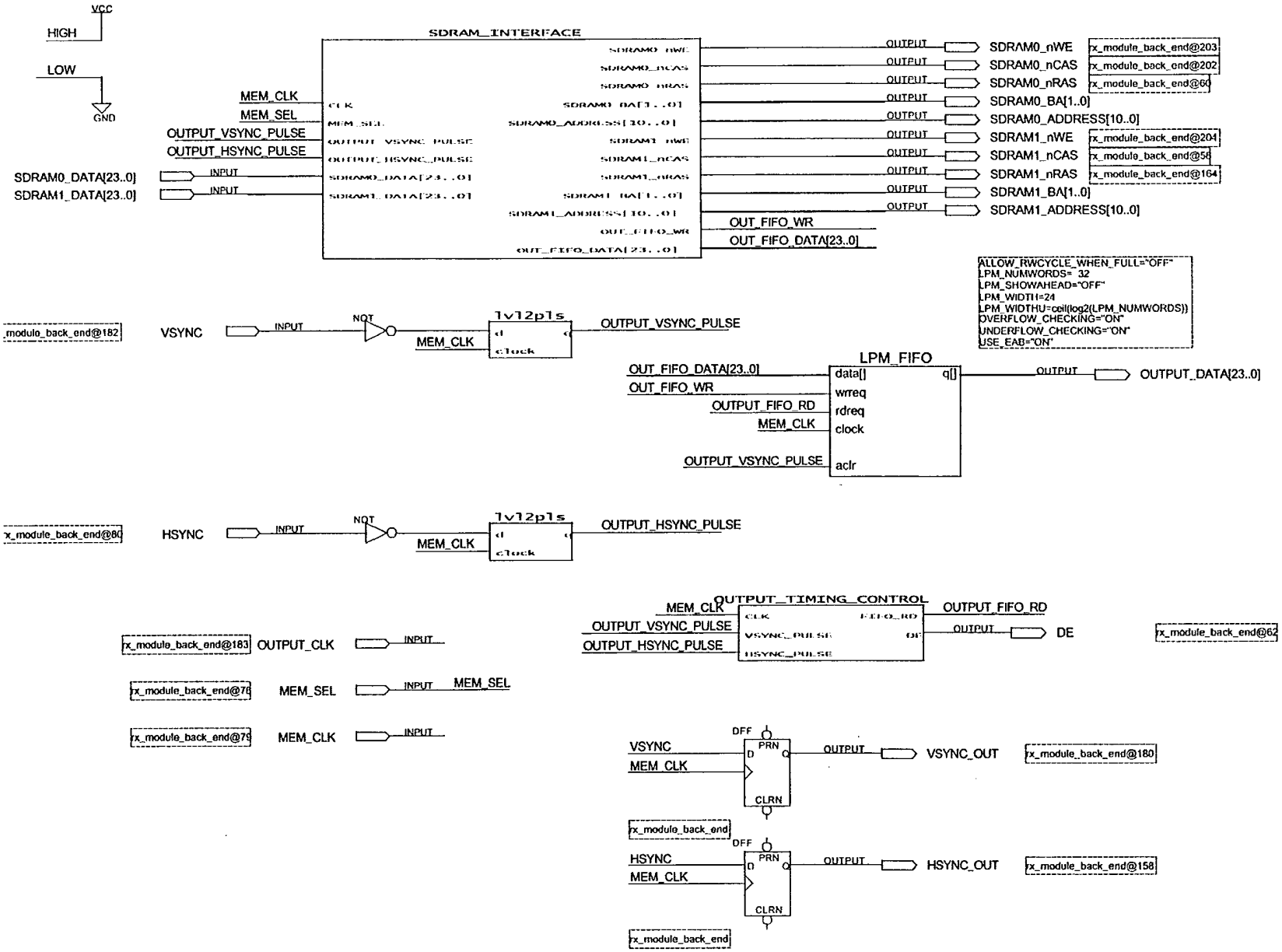
```
                                        END IF;
                                END CASE;
        END CASE;
        END PROCESS SM;


        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
                IF CLK = '1' THEN
                        memory_ps <= memory_ns;
                        init_ps <= init_ns;
                        normal_op_ps <= normal_op_ns;
                END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

APPENDIX F

RECEIVER VERY HIGH-LEVEL DESCRIPTION LANGUAGE

The following are the VHDL source files for the ADI receiver PLDs. The top level diagrams for the front and back end interfaces are followed by the VHDL source for each major PLD block.


LPM_FIFO_DC
SDRAM_INTERFACE
LPM_COMPARE
LPM_CONSTANT
MEM_SWITCH
LPM_FF
MEM_CLK
INPUT_VSYNC_PULSE
IN_FIFO_RD
IN_FIFO_READY
MEM_SEL
NOT_FIRST_FRAME
VSYNC
HSYNC
VSYNC, HSYNC
OUTPUT_VSYNC_PULSE
OUTPUT_HSYNC_PULSE
SDRAM0_nWE
SDRAM0_nCAS
SDRAM0_nRAS
SDRAM0_BA[1..0]
SDRAM0_ADDRESS[10..0]
SDRAM1_nWE
SDRAM1_nCAS
SDRAM1_nRAS
SDRAM1_BA[1..0]
SDRAM1_ADDRESS[10..0]
SDRAM0_DATA[23..0]
SDRAM1_DATA[23..0]
RED[7..0]
GREEN[7..0]
BLUE[7..0]
RED[7..0], GREEN[7..0], BLUE[7..0]


VCC
HIGH
LOW
GND
SDRAM_INTERFACE
MEM_CLK
MEM_SEL
OUTPUT_VSYNC_PULSE
OUTPUT_HSYNC_PULSE
SDRAM0_DATA[23..0]
SDRAM1_DATA[23..0]
INPUT
SDRAM0_nWE
SDRAM0_nCAS
SDRAM0_nRAS
SDRAM0_BA[1..0]
SDRAM0_ADDRESS[10..0]
SDRAM1_nWE
SDRAM1_nCAS
SDRAM1_nRAS
SDRAM1_BA[1..0]
SDRAM1_ADDRESS[10..0]
OUT_FIFO_WR
OUT_FIFO_DATA[23..0]
ALLOW_RWCYCLE_WHEN_FULL="OFF"
LPM_NUMWORDS= 32
LPM_SHOWAHEAD="OFF"
LPM_WIDTH=24
LPM_WIDTHU=ceil(log2(LPM_NUMWORDS))
OVERFLOW_CHECKING="ON"
UNDERFLOW_CHECKING="ON"
USE_EAB="ON"
VSYNC
NOT
lv12pls
OUTPUT_VSYNC_PULSE
LPM_FIFO
data[]
wrreq
rdreq
clock
aclr
q[]
OUTPUT_FIFO_RD
OUTPUT_DATA[23..0]
HSYNC
OUTPUT_HSYNC_PULSE
OUTPUT_TIMING_CONTROL
OUTPUT_FIFO_RD
DE
OUTPUT_CLK
DFF
PRN
CLRN
VSYNC_OUT
HSYNC_OUT

SDRAM Input Memory Interface

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY SDRAM_INTERFACE IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                IN_FIFO_DATA: IN        STD_LOGIC_VECTOR(23 DOWNTO 0);
                MEM_SEL : IN STD_LOGIC;

                INPUT_VSYNC_PULSE : IN STD_LOGIC;
                IN_FIFO_READY : IN STD_LOGIC;

                SDRAM0_nWE, SDRAM0_nCAS, SDRAM0_nRAS : OUT STD_LOGIC;
                SDRAM0_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM0_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                SDRAM1_nWE, SDRAM1_nCAS, SDRAM1_nRAS : OUT STD_LOGIC;
                SDRAM1_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM1_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                IN_FIFO_RD : OUT STD_LOGIC;

                SDRAM0_DATA, SDRAM1_DATA : INOUT STD_LOGIC_VECTOR(23 DOWNTO 0)

        );
END SDRAM_INTERFACE ;

ARCHITECTURE ONE OF SDRAM_INTERFACE IS

        SIGNAL  in_fifo_rd_d : STD_LOGIC;
        SIGNAL  nMEM_SEL: STD_LOGIC;
        SIGNAL  sdram0_mem_signals, sdram1_mem_signals : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL  input_mem_signals, registered_input_mem_signals : STD_LOGIC_VECTOR(16 DOWNTO 0);
        SIGNAL  INPUT_nWE, INPUT_nCAS, INPUT_nRAS : STD_LOGIC;
        SIGNAL  INPUT_BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL  INPUT_ADDRESS : STD_LOGIC_VECTOR(10 DOWNTO 0);

        COMPONENT INPUT_MEM_CONTROL IS
        PORT
        (
                tp_column_en, tp_column_rdy: OUT STD_LOGIC;

                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                IN_FIFO_FULL : IN STD_LOGIC;

                IN_FIFO_RD : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
        );
        END COMPONENT;

BEGIN

        INPUT_CONTROL : INPUT_MEM_CONTROL
                PORT MAP(
                        CLK => CLK,
                        VSYNC_PULSE => INPUT_VSYNC_PULSE,
                        IN_FIFO_FULL => IN_FIFO_READY,
                        IN_FIFO_RD => in_fifo_rd_d,
```

```
                nWE => INPUT_nWE,
                nCAS => INPUT_nCAS,
                nRAS => INPUT_nRAS,
                BA => INPUT_BA,
                ADDRESS => INPUT_ADDRESS
        );

    nMEM_SEL <= NOT MEM_SEL;

    TRI_STATE_BUSSES:
    FOR i IN 0 to 23 GENERATE

        SDRAM0_DATA_BUS: TRI
        PORT MAP(a_in => IN_FIFO_DATA(i), oe => nMEM_SEL, a_out => SDRAM0_DATA(i));

        SDRAM1_DATA_BUS: TRI
        PORT MAP(a_in => IN_FIFO_DATA(i), oe => MEM_SEL, a_out => SDRAM1_DATA(i));
    END GENERATE;

    input_mem_signals <=
        in_fifo_rd_d &
        INPUT_nWE &
        INPUT_nCAS &
        INPUT_nRAS &
        INPUT_BA(1 DOWNTO 0) &
        INPUT_ADDRESS(10 DOWNTO 0);

    SDRAM0_nWE <= sdram0_mem_signals(15);
    SDRAM0_nCAS <= sdram0_mem_signals(14);
    SDRAM0_nRAS <= sdram0_mem_signals(13);
    SDRAM0_BA(1 DOWNTO 0) <= sdram0_mem_signals(12 DOWNTO 11);
    SDRAM0_ADDRESS(10 DOWNTO 0) <= sdram0_mem_signals(10 DOWNTO 0);

    SDRAM1_nWE <= sdram1_mem_signals(15);
    SDRAM1_nCAS <= sdram1_mem_signals(14);
    SDRAM1_nRAS <= sdram1_mem_signals(13);
    SDRAM1_BA(1 DOWNTO 0) <= sdram1_mem_signals(12 DOWNTO 11);
    SDRAM1_ADDRESS(10 DOWNTO 0) <= sdram1_mem_signals(10 DOWNTO 0);

    INPUT_REGISTER : LPM_FF
        GENERIC MAP(LPM_WIDTH => 17)
        PORT MAP(data => input_mem_signals, clock => CLK, q =>
registered_input_mem_signals);

    IN_FIFO_RD <= registered_input_mem_signals(16);

    CONTROL_BUSSES:
    FOR i IN 0 to 15 GENERATE
        SDRAM0_CONTROL_BUS: TRI
        PORT MAP(a_in => registered_input_mem_signals(i), oe => nMEM_SEL, a_out =>
sdram0_mem_signals(i));

        SDRAM1_CONTROL_BUS: TRI
        PORT MAP(a_in => registered_input_mem_signals(i), oe => MEM_SEL, a_out =>
sdram1_mem_signals(i));
    END GENERATE;

END ONE;
```

ADI Memory Switch

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY MEM_SWITCH IS
    PORT
    (
--          tp_hs_en, tp_vs_en: OUT STD_LOGIC;
```

```
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;

                SEL: OUT STD_LOGIC;
                NOT_FIRST_FRAME: OUT STD_LOGIC
        );
END MEM_SWITCH;

ARCHITECTURE ONE OF MEM_SWITCH IS

        SIGNAL output_vsync: STD_LOGIC;
        SIGNAL output_vsync_pulse : STD_LOGIC;

        SIGNAL sel_ff_data, sel_ff_q : STD_LOGIC_VECTOR(0 DOWNTO 0);

        TYPE state_type IS (remaining_vsyncs, first_vsync);
        SIGNAL ps, ns : state_type := remaining_vsyncs;

        COMPONENT lvl2pls
                PORT(
                        d               : IN    STD_LOGIC;
                        clock   : IN    STD_LOGIC;
                        q               : OUT   STD_LOGIC);
        END COMPONENT;

BEGIN

        VSYNC_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => VSYNC_PULSE, cout => output_vsync);

        VSYNC_LVL2PLS: lvl2pls
                PORT MAP(d => output_vsync, clock => CLK, q => output_vsync_pulse);

        MEM_SEL_FF: lpm_ff
                GENERIC MAP(LPM_WIDTH => 1)
                PORT MAP(clock => CLK, enable => output_vsync_pulse, data => sel_ff_data, q =>
sel_ff_q);

        sel_ff_data(0) <= NOT sel_ff_q(0);
        SEL <= sel_ff_q(0);

        PROCESS(ps, output_vsync_pulse, VSYNC_PULSE)
                BEGIN
                        CASE ps IS

                                WHEN remaining_vsyncs =>
                                        NOT_FIRST_FRAME <= '1';

                                        IF output_vsync_pulse = '1' THEN
                                                ns <= first_vsync;
                                        ELSE
                                                ns <= remaining_vsyncs;
                                        END IF;

                                WHEN first_vsync =>
                                        NOT_FIRST_FRAME <= '0';

                                        IF VSYNC_PULSE = '1' THEN
                                                ns <= remaining_vsyncs;
                                        ELSE
                                                ns <= first_vsync;
                                        END IF;

                        END CASE;
        END PROCESS;

        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
```

```
            IF CLK = '1' THEN
                    ps <= ns;
            END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

SDRAM Output Memory Interface

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY SDRAM_INTERFACE IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                MEM_SEL : IN STD_LOGIC;

                OUTPUT_VSYNC_PULSE : IN STD_LOGIC;
                OUTPUT_HSYNC_PULSE : IN STD_LOGIC;

                SDRAM0_DATA, SDRAM1_DATA : IN STD_LOGIC_VECTOR(23 DOWNTO 0);

                SDRAM0_nWE, SDRAM0_nCAS, SDRAM0_nRAS : OUT STD_LOGIC;
                SDRAM0_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM0_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                SDRAM1_nWE, SDRAM1_nCAS, SDRAM1_nRAS : OUT STD_LOGIC;
                SDRAM1_BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                SDRAM1_ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);

                OUT_FIFO_WR : OUT STD_LOGIC;
                OUT_FIFO_DATA : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
        );
END SDRAM_INTERFACE;

ARCHITECTURE ONE OF SDRAM_INTERFACE IS

        SIGNAL  nMEM_SEL: STD_LOGIC;
        SIGNAL  OUT_FIFO_WR_PIPE_IN : STD_LOGIC;
        SIGNAL  MEM_SEL_VECTOR : STD_LOGIC_VECTOR(0 DOWNTO 0);

        SIGNAL  output_mem_signals, registered_output_mem_signals, sdram0_mem_signals,
sdram1_mem_signals : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL  OUTPUT_nWE, OUTPUT_nCAS, OUTPUT_nRAS : STD_LOGIC;
        SIGNAL  OUTPUT_BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL  OUTPUT_ADDRESS : STD_LOGIC_VECTOR(10 DOWNTO 0);
        SIGNAL   SDRAM_DATA : STD_LOGIC_2D(1 DOWNTO 0, 23 DOWNTO 0);

        COMPONENT OUTPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                HSYNC_PULSE : IN STD_LOGIC;

                OUT_FIFO_WR : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
        );
        END COMPONENT;
```

```
BEGIN

        OUPUT_CONTROL : OUTPUT_MEM_CONTROL
                PORT MAP(
                        CLK => CLK,
                        VSYNC_PULSE => OUTPUT_VSYNC_PULSE,
                        HSYNC_PULSE => OUTPUT_HSYNC_PULSE,
                        OUT_FIFO_WR => OUT_FIFO_WR_PIPE_IN,
                        nWE => OUTPUT_nWE,
                        nCAS => OUTPUT_nCAS,
                        nRAS => OUTPUT_nRAS,
                        BA => OUTPUT_BA,
                        ADDRESS => OUTPUT_ADDRESS
                );

        nMEM_SEL <= NOT MEM_SEL;

        OUT_FIFO_WR_PIPE : LPM_SHIFTREG
                GENERIC MAP(LPM_WIDTH => 4)
                PORT MAP(shiftin => OUT_FIFO_WR_PIPE_IN, clock => CLK, shiftout => OUT_FIFO_WR);

        output_mem_signals <=
                OUTPUT_nWE &
                OUTPUT_nCAS &
                OUTPUT_nRAS &
                OUTPUT_BA(1 DOWNTO 0) &
                OUTPUT_ADDRESS(10 DOWNTO 0);

        INPUT_REGISTER : LPM_FF
                GENERIC MAP(LPM_WIDTH => 16)
                PORT MAP(data => output_mem_signals, clock => CLK, q =>
registered_output_mem_signals);

        CONTROL_BUSSES:
        FOR i IN 0 TO 15 GENERATE
                SDRAM0_CONTROL_BUS: TRI
                PORT MAP(a_in => registered_output_mem_signals(i), oe => nMEM_SEL, a_out =>
sdram0_mem_signals(i));

                SDRAM1_CONTROL_BUS: TRI
                PORT MAP(a_in => registered_output_mem_signals(i), oe => MEM_SEL, a_out =>
sdram1_mem_signals(i));
        END GENERATE;

        SDRAM0_nWE <= sdram0_mem_signals(15);
        SDRAM0_nCAS <= sdram0_mem_signals(14);
        SDRAM0_nRAS <= sdram0_mem_signals(13);
        SDRAM0_BA(1 DOWNTO 0) <= sdram0_mem_signals(12 DOWNTO 11);
        SDRAM0_ADDRESS(10 DOWNTO 0) <= sdram0_mem_signals(10 DOWNTO 0);

        SDRAM1_nWE <= sdram1_mem_signals(15);
        SDRAM1_nCAS <= sdram1_mem_signals(14);
        SDRAM1_nRAS <= sdram1_mem_signals(13);
        SDRAM1_BA(1 DOWNTO 0) <= sdram1_mem_signals(12 DOWNTO 11);
        SDRAM1_ADDRESS(10 DOWNTO 0) <= sdram1_mem_signals(10 DOWNTO 0);

        DATA_BUSSES:
        FOR i IN 0 to 23 GENERATE
                SDRAM_DATA(0, i) <= SDRAM0_DATA(i);
                SDRAM_DATA(1, i) <= SDRAM1_DATA(i);
        END GENERATE;

        MEM_SEL_VECTOR(0) <= MEM_SEL;

        OUT_FIFO_DATA_MUX : lpm_mux
                GENERIC MAP(LPM_WIDTH => 24, LPM_SIZE => 2, LPM_WIDTHS => 1, LPM_PIPELINE => 1)
                PORT MAP(data => SDRAM_DATA, clock => CLK, sel => MEM_SEL_VECTOR, result =>
OUT_FIFO_DATA);

END ONE;
```

Output Timing Control

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY OUTPUT_TIMING_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                HSYNC_PULSE : IN        STD_LOGIC;

                FIFO_RD: OUT STD_LOGIC;
                DE : OUT STD_LOGIC
        );
END OUTPUT_TIMING_CONTROL;

ARCHITECTURE ONE OF OUTPUT_TIMING_CONTROL IS

        TYPE state_type IS (
                wait_for_vsync,
                wait_for_vertical_active,
                wait_for_hsync,
                wait_for_horizontal_active,
                display_enable
        );

        SIGNAL ns : state_type := wait_for_vsync;
        SIGNAL ps : state_type := wait_for_vsync;

        SIGNAL pixel_count_en, line_complete: STD_LOGIC;
        SIGNAL pixel_count, pixels_per_line : STD_LOGIC_VECTOR(10 DOWNTO 0);

        SIGNAL frame_complete: STD_LOGIC;
        SIGNAL line_count : STD_LOGIC_VECTOR(9 DOWNTO 0);

        SIGNAL front_porch_lines : STD_LOGIC_VECTOR(9 DOWNTO 0);
        SIGNAL front_porch_pixels : STD_LOGIC_VECTOR(10 DOWNTO 0);

        SIGNAL vertical_active_rdy, horizontal_active_rdy : STD_LOGIC;

BEGIN

        PIXEL_COUNTER : lpm_counter
                GENERIC MAP(LPM_WIDTH => 11, LPM_DIRECTION => "UP")
                PORT MAP(clock => CLK, cnt_en => pixel_count_en, q => pixel_count, sclr =>
HSYNC_PULSE);

        PIXEL_COMPARE : lpm_compare
                GENERIC MAP(LPM_WIDTH => 11)
                PORT MAP(dataa => pixel_count, datab => pixels_per_line, aeb => line_complete);

        PIXELS_PER_LINE_CONSTANT : lpm_constant
                GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 1290)
                PORT MAP(result => pixels_per_line);

        H_FRONT_PORCH_COMPARE : lpm_compare
                GENERIC MAP(LPM_WIDTH => 11)
                PORT MAP(dataa => pixel_count, datab => front_porch_pixels, aeb =>
horizontal_active_rdy);

        H_FRONT_PORCH_CONSTANT : lpm_constant
```

```
		GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 10)
		PORT MAP(result => front_porch_pixels);

	LINE_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 10, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => HSYNC_PULSE, q => line_count, sclr =>
VSYNC_PULSE, cout => frame_complete);

	V_FRONT_PORCH_COMPARE : lpm_compare
		GENERIC MAP(LPM_WIDTH => 10)
		PORT MAP(dataa => line_count, datab => front_porch_lines, aeb =>
vertical_active_rdy);

	V_FRONT_PORCH_CONSTANT : lpm_constant
		GENERIC MAP(LPM_WIDTH => 10, LPM_CVALUE => 5)
		PORT MAP(result => front_porch_lines);

	SM:
	PROCESS(ps, HSYNC_PULSE, VSYNC_PULSE, vertical_active_rdy, horizontal_active_rdy,
line_complete, frame_complete)
	BEGIN
		IF VSYNC_PULSE = '1' THEN
			ns <= wait_for_vertical_active;
		ELSE
			CASE ps IS

				WHEN wait_for_vsync =>
					pixel_count_en <= '0';
					FIFO_RD <= '0';
					DE <= '0';

					ns <= wait_for_vsync;

				WHEN wait_for_vertical_active =>
					pixel_count_en <= '0';
					FIFO_RD <= '0';
					DE <= '0';

					IF vertical_active_rdy = '1' THEN
						ns <= wait_for_hsync;
					ELSE
						ns <= wait_for_vertical_active;
					END IF;

				WHEN wait_for_hsync =>
					pixel_count_en <= '0';
					FIFO_RD <= '0';
					DE <= '0';

					IF HSYNC_PULSE = '1' THEN
						ns <= wait_for_horizontal_active ;
					ELSE
						ns <= wait_for_hsync;
					END IF;

				WHEN wait_for_horizontal_active =>
					pixel_count_en <= '1';
					FIFO_RD <= '0';
					DE <= '0';

					IF horizontal_active_rdy = '1' THEN
						ns <= display_enable;
					ELSE
						ns <= wait_for_horizontal_active ;
					END IF;


				WHEN display_enable =>
					pixel_count_en <= '1';
					FIFO_RD <= '1';
					DE <= '1';
```

```
                                        IF line_complete = '1' THEN
                                                IF frame_complete = '1' THEN

                                                        ns <= wait_for_vsync;
                                                ELSE
                                                        ns <= wait_for_hsync;
                                                END IF;
                                        ELSE
                                                ns <= display_enable ;
                                        END IF;

                        END CASE;
                END IF;
        END PROCESS SM;


        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
                IF CLK = '1' THEN
                        ps <= ns;
                END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

ADI Input Memory Controller

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY INPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                IN_FIFO_FULL : IN STD_LOGIC;

                IN_FIFO_RD : OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)

        );
END INPUT_MEM_CONTROL;

ARCHITECTURE ONE OF INPUT_MEM_CONTROL IS
        CONSTANT NUM_REFRESH_CYCLES: STD_LOGIC_VECTOR := "111";

        SIGNAL refresh_en, refresh_load, refresh_rdy : STD_LOGIC;
        SIGNAL refresh_cnt : STD_LOGIC_VECTOR(2 downto 0);

        TYPE init_state_type IS (start, wait_for_frame, precharge, refresh_nop, auto_refresh1,
wait_for_refresh1, auto_refresh2, wait_for_refresh2, load_mode_reg, done);
        SIGNAL init_ps, init_ns : init_state_type := start;

        TYPE memory_state_type IS (initialize, normal_operation);
        SIGNAL memory_ps, memory_ns : memory_state_type := initialize;
```

```
	TYPE normal_op_state_type IS (wait_for_FIFO, active, pre_write_nop, write_first_word,
burst, last_word, burst_terminate, precharge_nop1, precharge_nop2, precharge_nop3, refresh,
refresh_nop, auto_refresh, wait_for_refresh);
	SIGNAL normal_op_ps, normal_op_ns : normal_op_state_type := wait_for_FIFO;

	TYPE sdram_cmd_type IS (nop, active, read, write, precharge, refresh, load_mode_reg,
burst_terminate);
	SIGNAL sdram_cmd : sdram_cmd_type;
	SIGNAL test : std_logic_vector(2 DOWNTO 0);

	SIGNAL row_rdy, column_rdy, column_en, row_en, bank_en : STD_LOGIC;
	SIGNAL bank_count : STD_LOGIC_VECTOR(1 DOWNTO 0);
	SIGNAL row_count, row_count_ready : STD_LOGIC_VECTOR(10 DOWNTO 0);
	SIGNAL column_count : STD_LOGIC_VECTOR(7 DOWNTO 0);

	SIGNAL word_count_en, word_count_clr, word_count_rdy : STD_LOGIC;

BEGIN

	nRAS <= test(2);
	nCAS <= test(1);
	nWE <= test(0);

	WITH sdram_cmd SELECT
		test <= "0111" WHEN nop,
				"011" WHEN active,
				"101" WHEN read,
				"100" WHEN write,
				"010" WHEN precharge,
				"001" WHEN refresh,
				"110" WHEN burst_terminate,
				"000" WHEN load_mode_reg;

	AUTO_REFRESH_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 3, LPM_DIRECTION => "DOWN")
		PORT MAP(data => refresh_cnt, clock => CLK, cnt_en => refresh_en, sload =>
refresh_load, cout => refresh_rdy);

	COLUMN_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 8, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => column_en, q => column_count, sclr =>
VSYNC_PULSE, cout => column_rdy);

	ROW_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 11, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => row_en, q => row_count, sclr => VSYNC_PULSE);

	ROW_COMPARE : lpm_compare
		GENERIC MAP(LPM_WIDTH => 11)
		PORT MAP(dataa => row_count, datab => row_count_ready, aeb => row_rdy);

	ROW_COUNT_READY_CONSTANT : lpm_constant
		GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 1280)
		PORT MAP(result => row_count_ready);

	BANK_COUNTER: lpm_counter
		GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => bank_en, q => bank_count, sclr => VSYNC_PULSE);

	WORD_COUNTER: lpm_counter
		GENERIC MAP(LPM_WIDTH => 1, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => word_count_en, sclr => word_count_clr, cout =>
word_count_rdy);

	SM:
	PROCESS(memory_ps, init_ps, normal_op_ps, VSYNC_PULSE, refresh_rdy)
	BEGIN
		CASE memory_ps IS
			WHEN initialize =>
				IN_FIFO_RD <= '0';
				normal_op_ns <= wait_for_FIFO;
```

```
column_en <= '0';
row_en <= '0';
bank_en <= '0';
word_count_en <= '0';
word_count_clr <= '0';

CASE init_ps IS
        WHEN start=>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                IF VSYNC_PULSE= '1' THEN
                        init_ns <= wait_for_frame;
                ELSE
                        init_ns <= start;
                END IF;
        WHEN wait_for_frame=>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                IF VSYNC_PULSE= '1' THEN
                        init_ns <= precharge;
                ELSE
                        init_ns <= wait_for_frame;
                END IF;

        WHEN precharge=>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= precharge;
                ADDRESS <= "10000000000";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= refresh_nop;

        WHEN refresh_nop=>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= auto_refresh1;

        WHEN auto_refresh1=>
                refresh_cnt <= NUM_REFRESH_CYCLES;
                refresh_en <= '0';
                refresh_load <= '1';
                sdram_cmd <= refresh;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= wait_for_refresh1;
        WHEN wait_for_refresh1=>
                refresh_cnt <= "000";
                refresh_en <= '1';
```

```
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        IF refresh_rdy= '1' THEN
                                init_ns <= auto_refresh2;
                        ELSE
                                init_ns <= wait_for_refresh1;
                        END IF;
                WHEN auto_refresh2=>
                        refresh_cnt <= NUM_REFRESH_CYCLES;
                        refresh_en <= '0';
                        refresh_load <= '1';
                        sdram_cmd <= refresh;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        init_ns <= wait_for_refresh2;
                WHEN wait_for_refresh2=>
                        refresh_cnt <= "000";
                        refresh_en <= '1';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        memory_ns <= initialize;
                        IF refresh_rdy= '1' THEN
                                init_ns <= load_mode_reg;
                        ELSE
                                init_ns <= wait_for_refresh2;
                        END IF;

                WHEN load_mode_reg=>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= load_mode_reg;
                        ADDRESS <= "00000110111";
                        BA <= "00";

                        memory_ns <= initialize;
                        init_ns <= done;

                WHEN done=>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";

                        init_ns <= done;
                        memory_ns <= normal_operation;
        END CASE;

WHEN normal_operation =>

        memory_ns <= normal_operation;
        init_ns <= done;

        CASE normal_op_ps IS
                WHEN wait_for_FIFO =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
```

```
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '1';
        IN_FIFO_RD <= '0';

        IF IN_FIFO_FULL = '1' THEN
               normal_op_ns <= active;
        ELSE
               normal_op_ns <= wait_for_FIFO;
        END IF;

WHEN active =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= active;
        ADDRESS <= row_count;
        BA <= bank_count;
        IN_FIFO_RD <= '1';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= pre_write_nop;

WHEN pre_write_nop =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= write_first_word;


WHEN write_first_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= write;
        ADDRESS <= "10000000000";
        BA <= bank_count;
        IN_FIFO_RD <= '1';
        column_en <= '1';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';
        word_count_clr <= '0';

        normal_op_ns <= burst;

WHEN burst =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '1';
```

```
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';
        word_count_clr <= '0';

        IF column_rdy = '1' THEN
                IN_FIFO_RD <= '0';
                normal_op_ns <= last_word;
        ELSE
                IN_FIFO_RD <= NOT word_count_rdy;
                normal_op_ns <= burst;
        END IF;

WHEN last_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        word_count_clr <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '1';
        IN_FIFO_RD <= '0';

        normal_op_ns <= burst_terminate;

WHEN burst_terminate =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= burst_terminate;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        word_count_clr <= '0';
        column_en <= '0';
        row_en <= '1';
        bank_en <= row_rdy;
        word_count_en <= '1';

        normal_op_ns <= precharge_nop1 ;

WHEN precharge_nop1 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= precharge_nop2;

WHEN precharge_nop2 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
```

```
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= precharge_nop3;

WHEN precharge_nop3 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= refresh;

WHEN refresh=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= refresh_nop;

WHEN refresh_nop=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= auto_refresh;

WHEN auto_refresh =>
        refresh_cnt <= NUM_REFRESH_CYCLES;
        refresh_en <= '0';
        refresh_load <= '1';
        sdram_cmd <= refresh;
        ADDRESS <= "00000000000";
        BA <= "00";
        IN_FIFO_RD <= '0';
        column_en <= '0';
        row_en <= '0';
        bank_en <= '0';
        word_count_en <= '0';
        word_count_clr <= '0';

        normal_op_ns <= wait_for_refresh;

WHEN wait_for_refresh =>
```

```
                                        refresh_cnt <= "000";
                                        refresh_en <= '1';
                                        refresh_load <= '0';
                                        sdram_cmd <= nop;
                                        ADDRESS <= "00000000000";
                                        BA <= "00";
                                        IN_FIFO_RD <= '0';
                                        column_en <= '0';
                                        row_en <= '0';
                                        bank_en <= '0';
                                        word_count_en <= '0';
                                        word_count_clr <= '0';

                                        IF refresh_rdy= '1' THEN
                                                normal_op_ns <= wait_for_FIFO;
                                        ELSE
                                                normal_op_ns <= wait_for_refresh;
                                        END IF;

                                END CASE;
                        END CASE;
        END PROCESS SM;


        STATE_HANDLER:
        PROCESS(CLK)
        BEGIN
                IF CLK = '1' THEN
                        memory_ps <= memory_ns;
                        init_ps <= init_ns;
                        normal_op_ps <= normal_op_ns;
                END IF;
        END PROCESS STATE_HANDLER;

END ONE;
```

ADI Output Memory Controller

```
LIBRARY altera;
USE altera.maxplus2.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY OUTPUT_MEM_CONTROL IS
        PORT
        (
                CLK : IN        STD_LOGIC;
                VSYNC_PULSE : IN        STD_LOGIC;
                HSYNC_PULSE : IN        STD_LOGIC;

                OUT_FIFO_WR: OUT STD_LOGIC;
                nWE, nCAS, nRAS : OUT STD_LOGIC;
                BA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                ADDRESS : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)

        );
END OUTPUT_MEM_CONTROL;

ARCHITECTURE ONE OF OUTPUT_MEM_CONTROL IS
        CONSTANT NUM_REFRESH_CYCLES: STD_LOGIC_VECTOR := "111";

        SIGNAL refresh_en, refresh_load, refresh_rdy : STD_LOGIC;
        SIGNAL refresh_cnt : STD_LOGIC_VECTOR(2 downto 0);
```

```
	TYPE init_state_type IS (
		start,
		wait_for_frame,
		precharge,
		refresh_nop,
		auto_refresh1,
		wait_for_refresh1,
		auto_refresh2,
		wait_for_refresh2,
		load_mode_reg,
		done
	);

	SIGNAL init_ps, init_ns : init_state_type := start;

	TYPE memory_state_type IS (initialize, normal_operation);
	SIGNAL memory_ps, memory_ns : memory_state_type := initialize;

	TYPE normal_op_state_type IS (
		wait_for_vsync,
		wait_for_active_video,
		wait_for_hsync,
		active,
		pre_read_nop,
		read_first_word,
		burst,
		read_last_word,
		burst_terminate,
		precharge_nop1,
		precharge_nop2,
		precharge_nop3,
		refresh,
		refresh_nop,
		auto_refresh,
		wait_for_refresh
	);

	SIGNAL normal_op_ns : normal_op_state_type := wait_for_vsync;
	SIGNAL normal_op_ps : normal_op_state_type := wait_for_vsync;

	TYPE sdram_cmd_type IS (
		nop,
		active,
		read,
		write,
		precharge,
		refresh,
		load_mode_reg,
		burst_terminate
	);

	SIGNAL sdram_cmd : sdram_cmd_type;
	SIGNAL test : std_logic_vector(2 DOWNTO 0);

	SIGNAL row_en, row_rdy : STD_LOGIC;
	SIGNAL column_en, column_rdy : STD_LOGIC;
	SIGNAL bank_en, bank_rdy : STD_LOGIC;
	SIGNAL frame_en, frame_rdy : STD_LOGIC;
	SIGNAL line_en, line_rdy : STD_LOGIC;

	SIGNAL bank_count : STD_LOGIC_VECTOR(1 DOWNTO 0);
	SIGNAL row_count, row_count_ready : STD_LOGIC_VECTOR(10 DOWNTO 0);
	SIGNAL line_count, line_count_ready : STD_LOGIC_VECTOR(2 DOWNTO 0);
	SIGNAL column_count : STD_LOGIC_VECTOR(7 DOWNTO 0);

	SIGNAL front_porch_count, front_porch_lines : STD_LOGIC_VECTOR(3 DOWNTO 0);
	SIGNAL active_video_rdy : STD_LOGIC;

BEGIN
```

```
	nRAS <= test(2);
	nCAS <= test(1);
	nWE <= test(0);

	WITH sdram_cmd SELECT
		test <=
				"111" WHEN nop,
				"011" WHEN active,
				"101" WHEN read,
				"100" WHEN write,
				"010" WHEN precharge,
				"001" WHEN refresh,
				"110" WHEN burst_terminate,
				"000" WHEN load_mode_reg;


	AUTO_REFRESH_COUNTER : lpm_counter .
		GENERIC MAP(LPM_WIDTH => 3, LPM_DIRECTION => "DOWN")
		PORT MAP(data => refresh_cnt, clock => CLK, cnt_en => refresh_en, sload =>
refresh_load, cout => refresh_rdy);

	COLUMN_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 8, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => column_en, q => column_count, sclr =>
VSYNC_PULSE, cout => column_rdy);

	ROW_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 11, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => row_en, q => row_count, sclr => VSYNC_PULSE);

	ROW_COMPARE : lpm_compare
		GENERIC MAP(LPM_WIDTH => 11)
		PORT MAP(dataa => row_count, datab => row_count_ready, aeb => row_rdy);

	ROW_COUNT_READY_CONSTANT : lpm_constant
		GENERIC MAP(LPM_WIDTH => 11, LPM_CVALUE => 1280)
		PORT MAP(result => row_count_ready);

	LINE_COUNTER : lpm_counter
		GENERIC MAP(LPM_WIDTH => 3, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => line_en, q => line_count, sclr => HSYNC_PULSE);

	LINE_COMPARE : lpm_compare
		GENERIC MAP(LPM_WIDTH => 3)
		PORT MAP(dataa => line_count, datab => line_count_ready, aeb => line_rdy);

	LINE_COUNT_READY_CONSTANT : lpm_constant
		GENERIC MAP(LPM_WIDTH => 3, LPM_CVALUE => 5)
		PORT MAP(result => line_count_ready);

	BANK_COUNTER: lpm_counter
		GENERIC MAP(LPM_WIDTH => 2, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => bank_en, q => bank_count, sclr => VSYNC_PULSE,
cout => bank_rdy);

	FRAME_COUNTER: lpm_counter
		GENERIC MAP(LPM_WIDTH => 1, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => frame_en, sclr => VSYNC_PULSE,  cout =>
frame_rdy);

	FRONT_PORCH_COUNTER: lpm_counter
		GENERIC MAP(LPM_WIDTH => 4, LPM_DIRECTION => "UP")
		PORT MAP(clock => CLK, cnt_en => HSYNC_PULSE, sclr => VSYNC_PULSE, q =>
front_porch_count);

	FRONT_PORCH_COMPARE : lpm_compare
		GENERIC MAP(LPM_WIDTH => 4)
		PORT MAP(dataa => front_porch_count, datab => front_porch_lines, aeb =>
active_video_rdy);

	FRONT_PORCH_LINES_CONSTANT : lpm_constant
```

```
            GENERIC MAP(LPM_WIDTH => 4, LPM_CVALUE => 10)
            PORT MAP(result => front_porch_lines);

        SM:
        PROCESS(memory_ps, init_ps, normal_op_ps, HSYNC_PULSE, VSYNC_PULSE, refresh_rdy,
column_rdy, frame_rdy, active_video_rdy)
        BEGIN
                CASE memory_ps IS
                        WHEN initialize =>
                                OUT_FIFO_WR <= '0';
                                normal_op_ns <= wait_for_vsync;
                                column_en <= '0';
                                row_en <= '0';
                                line_en <= '0';
                                bank_en <= '0';
                                frame_en <= '0';

                                CASE init_ps IS
                                        WHEN start =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
                                                sdram_cmd <= nop;
                                                ADDRESS <= "00000000000";
                                                BA <= "00";

                                                memory_ns <= initialize;
                                                IF VSYNC_PULSE= '1' THEN
                                                        init_ns <= wait_for_frame;
                                                ELSE
                                                        init_ns <= start;
                                                END IF;

                                        WHEN wait_for_frame =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
                                                sdram_cmd <= nop;
                                                ADDRESS <= "00000000000";
                                                BA <= "00";

                                                memory_ns <= initialize;
                                                IF VSYNC_PULSE= '1' THEN
                                                        init_ns <= precharge;
                                                ELSE
                                                        init_ns <= wait_for_frame;
                                                END IF;

                                        WHEN precharge =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
                                                sdram_cmd <= precharge;
                                                ADDRESS <= "10000000000";
                                                BA <= "00";

                                                memory_ns <= initialize;
                                                init_ns <= refresh_nop;

                                        WHEN refresh_nop =>
                                                refresh_cnt <= "000";
                                                refresh_en <= '0';
                                                refresh_load <= '0';
                                                sdram_cmd <= nop;
                                                ADDRESS <= "00000000000";
                                                BA <= "00";

                                                memory_ns <= initialize;
                                                init_ns <= auto_refresh1;

                                        WHEN auto_refresh1 =>
```

```
                refresh_cnt <= NUM_REFRESH_CYCLES;
                refresh_en <= '0';
                refresh_load <= '1';
                sdram_cmd <= refresh;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= wait_for_refresh1;
        WHEN wait_for_refresh1 =>
                refresh_cnt <= "000";
                refresh_en <= '1';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                IF refresh_rdy= '1' THEN
                        init_ns <= auto_refresh2;
                ELSE
                        init_ns <= wait_for_refresh1;
                END IF;
        WHEN auto_refresh2 =>
                refresh_cnt <= NUM_REFRESH_CYCLES;
                refresh_en <= '0';
                refresh_load <= '1';
                sdram_cmd <= refresh;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= wait_for_refresh2;
        WHEN wait_for_refresh2 =>
                refresh_cnt <= "000";
                refresh_en <= '1';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                memory_ns <= initialize;
                IF refresh_rdy= '1' THEN
                        init_ns <= load_mode_reg;
                ELSE
                        init_ns <= wait_for_refresh2;
                END IF;

        WHEN load_mode_reg =>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= load_mode_reg;
                ADDRESS <= "00000110111";
                BA <= "00";

                memory_ns <= initialize;
                init_ns <= done;

        WHEN done =>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";

                init_ns <= done;
                memory_ns <= normal_operation;
END CASE;
```

```
WHEN normal_operation =>
        memory_ns <= normal_operation;
        init_ns <= done;


        CASE normal_op_ps IS
                WHEN wait_for_active_video =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";
                        column_en <= '0';
                        row_en <= '0';
                        line_en <= '0';
                        bank_en <= '0';
                        frame_en <= '0';
                        OUT_FIFO_WR <= '0';

                        IF active_video_rdy = '1' THEN
                                normal_op_ns <= wait_for_hsync;
                        ELSE
                                normal_op_ns <= wait_for_active_video;
                        END IF;

                WHEN wait_for_hsync=>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";
                        column_en <= '0';
                        row_en <= '0';
                        line_en <= '0';
                        bank_en <= '0';
                        frame_en <= '0';
                        OUT_FIFO_WR <= '0';

                        IF HSYNC_PULSE = '1' THEN
                                normal_op_ns <= active;
                        ELSE
                                normal_op_ns <= wait_for_hsync;
                        END IF;

                WHEN active =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= active;
                        ADDRESS <= row_count;
                        BA <= bank_count;
                        column_en <= '0';
                        row_en <= '0';
                        line_en <= '0';
                        bank_en <= '0';
                        frame_en <= '0';
                        OUT_FIFO_WR <= '0';

                        normal_op_ns <= pre_read_nop;

                WHEN pre_read_nop =>
                        refresh_cnt <= "000";
                        refresh_en <= '0';
                        refresh_load <= '0';
                        sdram_cmd <= nop;
                        ADDRESS <= "00000000000";
                        BA <= "00";
                        column_en <= '1';
                        row_en <= '0';
```

```
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= read_first_word;

WHEN read_first_word =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= read;
        ADDRESS <= "10000000000";
        BA <= bank_count;
        column_en <= '1';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= burst;

WHEN burst =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '1';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '1';

        IF column_rdy = '1' THEN
                normal_op_ns <= read_last_word;
        ELSE
                normal_op_ns <= burst;
        END IF;

WHEN read_last_word=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '1';

        normal_op_ns <= burst_terminate;

WHEN burst_terminate =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= burst_terminate;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '1';
        line_en <= '1';
        bank_en <= row_rdy;
        frame_en <= bank_rdy AND row_rdy;
```

```
        OUT_FIFO_WR <= '1';

        normal_op_ns <= precharge_nop1;

WHEN precharge_nop1 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= precharge_nop2;

WHEN precharge_nop2 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= precharge_nop3;

WHEN precharge_nop3 =>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= nop;
        ADDRESS <= "00000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= refresh;

WHEN refresh=>
        refresh_cnt <= "000";
        refresh_en <= '0';
        refresh_load <= '0';
        sdram_cmd <= precharge;
        ADDRESS <= "10000000000";
        BA <= "00";
        column_en <= '0';
        row_en <= '0';
        line_en <= '0';
        bank_en <= '0';
        frame_en <= '0';
        OUT_FIFO_WR <= '0';

        normal_op_ns <= refresh_nop;

WHEN refresh_nop=>
        refresh_cnt <= "000";
        refresh_en <= '0';
```

```
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";
                column_en <= '0';
                row_en <= '0';
                line_en <= '0';
                bank_en <= '0';
                frame_en <= '0';
                OUT_FIFO_WR <= '0';

                normal_op_ns <= auto_refresh;

        WHEN auto_refresh =>
                refresh_cnt <= NUM_REFRESH_CYCLES;
                refresh_en <= '0';
                refresh_load <= '1';
                sdram_cmd <= refresh;
                ADDRESS <= "00000000000";
                BA <= "00";
                column_en <= '0';
                row_en <= '0';
                line_en <= '0';
                bank_en <= '0';
                frame_en <= '0';
                OUT_FIFO_WR <= '0';

                normal_op_ns <= wait_for_refresh;

        WHEN wait_for_refresh =>
                refresh_cnt <= "000";
                refresh_en <= '1';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";
                column_en <= '0';
                row_en <= '0';
                line_en <= '0';
                bank_en <= '0';
                frame_en <= '0';
                OUT_FIFO_WR <= '0';

                IF refresh_rdy= '1' THEN
                        IF frame_rdy = '1' THEN
                                normal_op_ns <= wait_for_vsync;
                        ELSE
                                IF line_rdy = '1' THEN
                                        normal_op_ns <=
wait_for_hsync;
                                ELSE
                                        normal_op_ns <= active;
                                END IF;
                        END IF;
                ELSE
                        normal_op_ns <= wait_for_refresh;
                END IF;

        WHEN wait_for_vsync =>
                refresh_cnt <= "000";
                refresh_en <= '0';
                refresh_load <= '0';
                sdram_cmd <= nop;
                ADDRESS <= "00000000000";
                BA <= "00";
                column_en <= '0';
                row_en <= '0';
                line_en <= '0';
                bank_en <= '0';
                frame_en <= '0';
                OUT_FIFO_WR <= '0';
```

```
                                        IF VSYNC_PULSE = '1' THEN
                                                normal_op_ns <= wait_for_active_video;
                                        ELSE
                                                normal_op_ns <= wait_for_vsync;
                                        END IF;
                                END CASE;
        END CASE;
    END PROCESS SM;


    STATE_HANDLER:
    PROCESS(CLK)
    BEGIN
        IF CLK = '1' THEN
            memory_ps <= memory_ns;
            init_ps <= init_ns;
            normal_op_ps <= normal_op_ns;
        END IF;
    END PROCESS STATE_HANDLER;

END ONE;
```