A Synchronous Distributed Digital Control Architecture for High Power Converters

Gerald Francis

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

Master of Science
in
Electrical Engineering

Dushan Boroyevich, Chair
Pushkin Kachroo
William Tranter

March 3rd, 2004
Blacksburg, Virginia

Keywords: Communication, Control, Digital Systems, Distributed Systems, DSP, Fiber Optics, FPGA, PEBB, Plug and Play, Power Electronics

A Synchronous Distributed Digital Control Architecture for High Power Converters

Gerald Francis

(ABSTRACT)

Power electronics applications in high power are normally large, expensive, spatially distributed systems. These systems are typically complex and have multiple functions. Due to these properties, the control algorithm and its implementation are challenging, and a different approach is needed to avoid customized solutions to every application while still having reliable sensor measurements and converter communication and control.

This thesis proposes a synchronous digital control architecture that allows for the communication and control of devices via a fiber optic communication ring using digital technology. The proposed control architecture is a multidisciplinary approach consisting of concepts from several areas of electrical engineering. A review of the state of the art is presented in Chapter 2 in the areas of power electronics, fieldbus control networks, and digital design. A universal controller is proposed as a solution to the hardware independent control of these converters. Chapter 3 discusses how the controller was specified, designed, implemented, and tested. The power level specific hardware is implemented in modules referred to as hardware managers. A design for a hardware manager was previously implemented and tested. Based on these results and experiences, an improved hardware manager is specified in Chapter 4. A fault tolerant communication protocol is specified in Chapter 5. This protocol is an improvement on a previous version of the protocol, adding benefits of improved synchronization, multimaster support, fault tolerant structure with support for hot-swapping, live insertion and removals, a variable ring structure, and a new network based clock concept for greater flexibility and control. Chapter 6 provides a system demonstration, verifying the components work in configurations involving combinations of controllers and hardware managers to form applications. Chapter 7 is the conclusion. VHDL code is included for the controller, the hardware manager, and the protocol. Schematics and manufacturing specifications are included for the controller.

Grant Information

This work was supported by grants from the Office of Naval Research (ONR) under award number N00014-00-1-0489[1] and award number N00014-03-1-0771[2].

This work made use of ERC shared facilities supported by the National Science Foundation (NSF) under award number EEC-9731677[3].

[1] Power Electronics Building Blocks Plug and Play Hardware and Software Control Architectures

[2] Standard Cell Open Architecture Power Conversion Systems

[3] Center for Power Electronics Systems

Dedication

To my mother, Margaret Francis,
and to my grandparents, Alice and Calvin Clingan

Acknowledgements

I would like to thank my ONR project team members and the co-principal investigators for providing very useful insight and feedback: Dr. Fred Wang, Dr. Stephen Edwards, Dr. Rolando Burgos, Mr. Daniel Ghizoni, Dr. Jinghong Guo, Mr. Sebastian Rosado, Mr. Kuljeet Singh, Ms. Sumithra Bhakthavatsalam, Ms. Parool Mody, Mr. Christopher Einsmann, and Mr. Jay Gentry. I would also like to thank Dr. Yunfeng Liu, Dr. Zhenxue Xu, Dr. Bin Zhang, and Dr. Siriroj Sirisukprasert for their valuable conversations on the application of this architecture to higher power systems.

I would like to thank my friends and coworkers for their support: Mr. Aaron Downey, Mr. Anthony Miller, Ms. Marina Spanos, Mr. Elton Pepa, Mrs. Carrie L. Pepa, Mr. William Cornwell, Mrs. Allison S. Cornwell, Mr. Christopher Smith, Mr. Brett Waldo, Ms. Michelle Kosmo, Mr. Michael Gilliom, Mr. Justin Pinckney, Mr. Bryan Charboneau, Mr. Tim Thacker, Mr. Carson Baisden, Mr. Joel Gouker, Ms. Amy Johnson, and many others, all of whom would be impossible to list.

I would like to thank the CPES directors and staff for their support: Dr. Fred Lee, Dr. Elizabeth Tranter, Mrs. Michelle Czamenske, Mrs. Teresa Shaw, Mrs. Marianne Hawthorne, Mr. David Fuller, Mrs. Linda Long, Mrs. Linda Gallagher, Mrs. Ann Craig, Mrs. Teresa Rose, Mr. Robert Martin, Mr. Jamie Evans, Mr. Steve Chen, Mr. Ben Pfountz, Mr. Matt Pfountz, Mr. Rob Chang, Mr. Gary Kerr, Mr. Mike King, and Mrs. Leslie Farmer. I would also like to thank Tektronix for their large donations that enabled us to use time saving, high end state of the art equipment to conduct this research.

Many problems were solved and issues discussed over many teleconferences with General Dynamics Advanced Information Systems. I would like to specifically thank Mr. Olav Grip, Mr. Tom Brennan, Mr. Dan Fabiano, and Mr. Steven Baum for all the time, insight, suggestions, and support provided. I would also like to thank the members of other organizations for their time in review meetings from General Dynamics Advanced Information Systems, Northrop Grumman PCS, Northrop Grumman Newport News Shipyard, Bechtel Bettis Atomic Power Labs, Kaman Aerospace, Naval Surface Warfare Center Carderock Division, ABB, Rockwell, Silicon Power, and others.

I would like to thank my committee members Dr. Pushkin Kachroo and Dr. William Tranter, and project sponsor, Mr. Terry Ericsen of the Office of Naval Research for his inspiring vision of future naval capabilities and for providing the funds necessary to achieve the goals described in this thesis.

I would especially like to thank my advisor, Dr. Dushan Boroyevich for all of his insight, guidance, feedback, support, and trust. Without him, none of this would be possible.

Finally and most importantly, I would like to thank my Mother, Sister, and Grandparents. I would have never been here without their wonderful support and love.

Table of Contents

Table of Figures

Table of Tables

1 Introduction and Background

This thesis introduces a system architecture that enables a modular design of power conversion systems in high power applications. Power converter applications for higher power systems have challenges associated with the converter's control, application, and location/distribution. The application may add more requirements such as survivability and fault tolerance. In high power systems, the costs of devices and supporting framework are much higher than the costs incurred in lower power systems, which in turn justifies a more advanced and flexible communications and control architecture, despite the added cost.

1.1 Requirements of High Power Converters

Many power converters are not stand-alone systems. They exist within a system that is working together with other components to create an application. Several power converter applications that may require the power system to have additional intelligence include:

- Systems with advanced protection
- Systems with observers or complex control algorithms [1, 2]
- Systems with dynamic reconfiguration capability
- Systems with monitoring and status reporting
- Spatially distributed systems

These systems support applications that may be commercial, industrial, or military. Examples of these applications are:

- Pulsed power systems and Electric Weapons [3]
- Highly available power conversion systems [5]
- Terrestrial power distribution systems (STATCOM, DVR, UPFC, other FACTS controllers) [4, 5]
- Shipboard Power and Propulsion Systems [6, 7]
- Large industrial drives [8, 9]

- Aircraft Power Systems
- Electric Trains
- Industrial Power Distribution Systems

Reconfiguration. Many survivable systems are moving towards reconfiguration as a means to adapt to changing environmental conditions and mission requirements [10]. Reconfiguration adds special requirements for advanced communication and control systems, such as the ability to dynamically change the function or service that the specific converter provides as requested by another part of the system. Reconfigurable power systems can address issues such as environments that can change their load requirements at any given time [3]. Similar research has been done in reconfigurable computing [11], where devices can change their configuration depending on the service that that component is to provide at the given time.

Therefore, to construct a power conversion system, the requirements of the system must be supported by a power electronics control architecture that allows the system to meet the requirements of the application or mission. Hence, the design of the individual components in the power conversion system will need to change to support these requirements.

Power converters are complex systems, especially as the power level increases. Depending on the application, it may have advanced features such as diagnostics and user interfaces. It may also participate on a control network. The aspects in designing a power converter are shown in Figure 1.

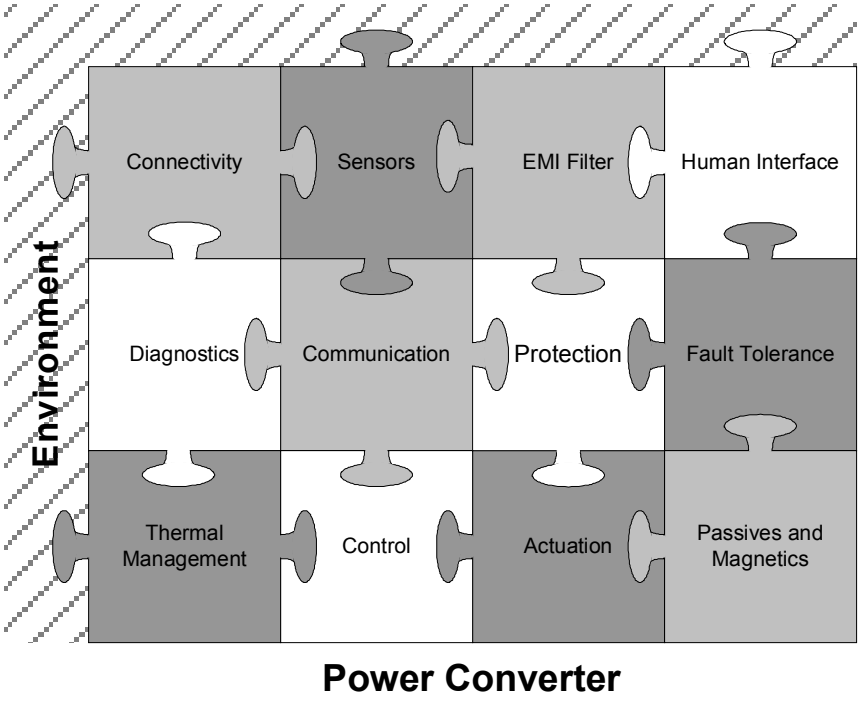


Figure 1: Power Converter Components

1.1.1 Power Electronics Building Blocks

Power Electronics Building Blocks (PEBB) is a modularized design concept that allows for power converters to be built using commercial off the shelf components [12]. These building blocks are for high power converters, which typically have power levels of 100kW and greater. The components are vendor independent implementations that can be connected together physically and via the control to form an application. Examples of PEBBs are phase legs, controllers, single switches and H-Bridges.

This approach has its advantages in changing the scope of power converter design from a holistic design process to an integration task. However, this approach implicitly adds design challenges to the module design and architecture design that are necessary to implement this type of system. This allows the designer to focus more on the application than on the details of the power converter.

1.1.2 Challenges in Controlling Power Electronics Building Blocks

The concept of modularization comes with some challenges that need to be solved. Traditional converter design is not appropriate for PEBB based systems. The modular, open architecture nature of PEBB systems imposes compatibility requirements on the system, as well as synchronization and interface issues. Similar issues have been encountered in the PC industry in the past as the PC became an open architecture system, allowing the individual components to be manufactured by different companies.

1.1.3 History of PEBB Based Converters

In the Center for Power Electronics Systems at Virginia Tech, several PEBB converters driving toward these goals have been created over the last 10 years. These converters are listed below.

- 4 Leg Inverter / Load Conditioner [13]
- Three Phase Matrix Converter [14]
- DC Bus Regulator [15]
- DC Bus Conditioner [16]
- SMES Power Conversion System (PCS) [17, 18, 19, 20]
- PEBB Based Inverter [66]

Control Signals. The 4-Leg inverter and the matrix converter used electrical signals to control the gate driver circuits. The DC bus conditioner, three-phase bus conditioner, and SMES PCS used optical signals to control the gating of the devices. The PEBB Inverter used a communication protocol to send duty cycle values to the inverter, which in turn, produced binary commands to each gate signal via an optically isolated input onboard the module.

The first five converters used a star topology. There were two direct connections between the controller and each switch: the switch command, and the fault signals. Similarly, analog variables being measured were brought to the controller, where they were sampled, and turned into digital signals. This structure is shown in Figure 2.

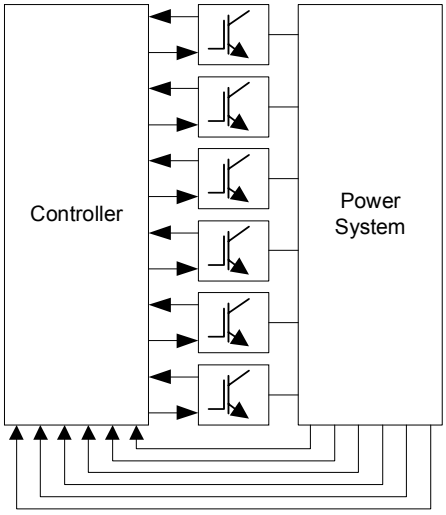


Figure 2: Star Topology

The sixth converter used a single fiber optic ring topology. This topology utilized a simple protocol to transmit data to each node, and then synchronized each node to make that data active. The structure of that topology is shown in Figure 3. The data transmitted from the controller was a duty cycle, and the data received from each slave node was the value of two sensors [21].

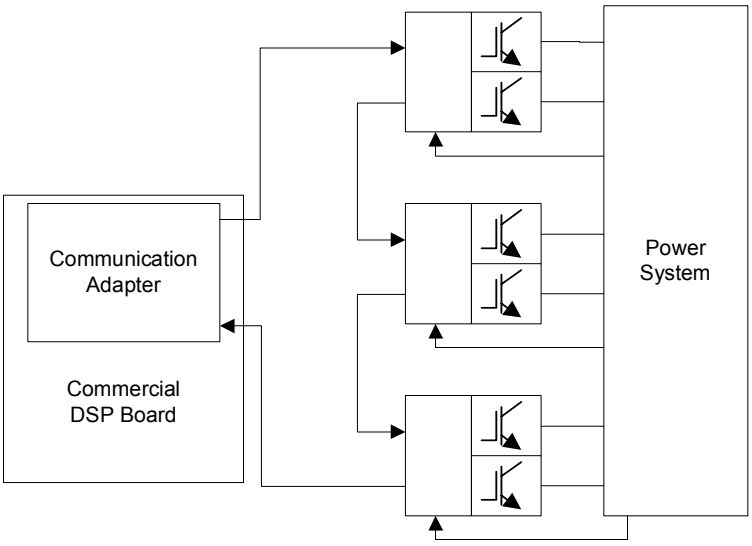


Figure 3: Ring Architecture

Table 1 summarizes these converters.

Table 1: PEBB Converter Summary

Converter	Switching Frequency	Control Signals	Power Level	Cooling
Matrix	10 kHz	Electrical Binary	30 kW	Air
4-Leg Inverter + Load Conditioner	5 kHz 20 kHz	Electrical Binary	150 kW 30 kW	Air
DC Bus Conditioner	20 kHz	Optical Binary		Water
DC Bus Conditioner	20 kHz	Optical Binary	100 kW	Water
SMES PCS	20 kHz	Optical Binary	250 kW	Water
PEBB Inverter	20 kHz	Optical Protocol	100 kW	Water

While all of these converters meet application requirement specifications, they are modular only in the physical aspect. The control is still coupled at a centralized location, and the control code is written to individually control and sense different components of the converter holistically and centrally.

This ties the implementation to the control, and simultaneously restricts the future expansion capability, and thus the scope of the converter's applications significantly. Major design cycles are required to adapt the converter's control algorithms and sensors to the application. The approaches, while modularized, are not plug and play. The effort required to build a converter from these components is still high, and remains unchanged at the control and sensor interface with the exception of the PEBB inverter.

The PEBB inverter is a step towards modularized control. However, there are issues implicit in the implementation that limit the scope of the application of this converter and architecture. The use of a single ring has negative impacts on reliability, and does not enable hot insertion or removal of nodes, or reconfiguration. The communication protocol is highly restricted in its capability, only allowing access to phase leg duty cycle and sensor commands, which limits the possible types of nodes that can be placed there,

their features, and thus the future expansion capability and flexibility of the control architecture.

1.2 Systems Approach

Modularization. Traditional power electronics design typically involves a ground-up design and implementation. Even though the design may be modular and even functionally partitioned, every time that a new design is required, it is necessary to manage non-standard interfaces that vary from design to design. The design process tends to be linear, terminating with the installation or release of the product, as shown in Figure 4.

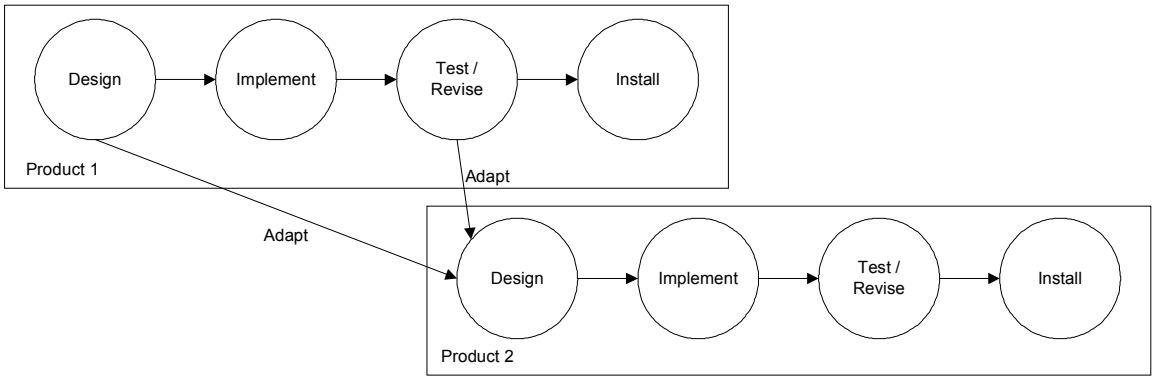


Figure 4: Traditional Design

In a modular design, it is possible to have parallel designs occurring at the same time. Once the modular design is complete, it may even be possible to replace defective modules with modules of the same type, similar to car parts. It is still, however, the responsibility of a specific manufacturer to design and implement the each of these modules. A modular design process can be illustrated as shown in Figure 5.

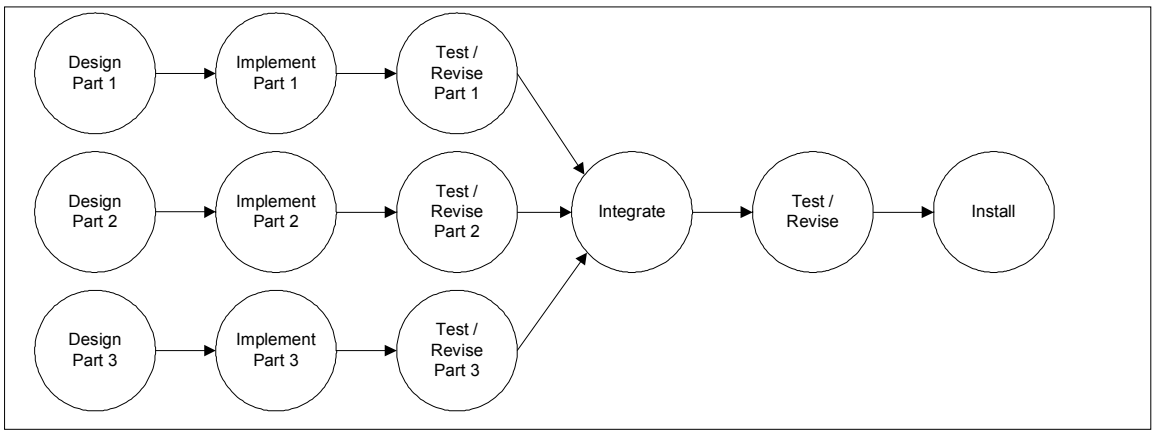


Figure 5: Modular Design

Ideally, the individual components should be able to undergo independent designs, where each component is designed and continually improved. These continuous improvements do not affect the design of the other modules, but enhance the behavior of the system. As pieces become available, they can be replace older versions of these pieces from possibly different vendors without a need to redesign the system or the control.

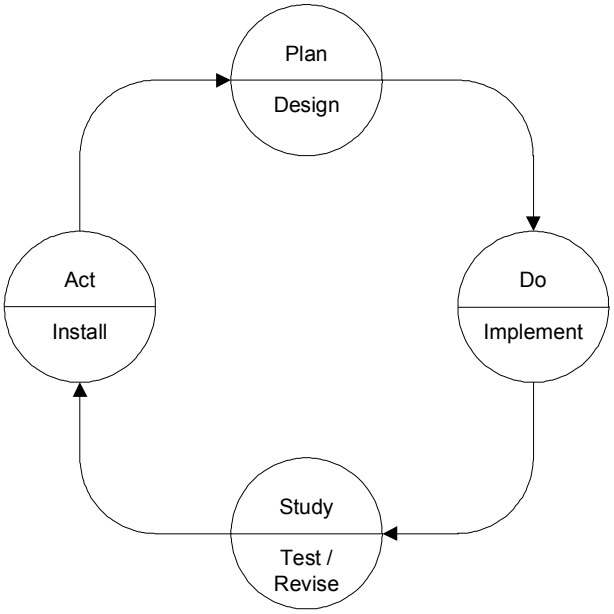


Figure 6: Continuous Improvement Process

In a plug and play design, it is possible to integrate different "modules" from different manufacturers to create a complete design. In order to implement this, it is necessary to have standardized interfaces and behaviors that allow compatible modules to be

interchangeable, such as the Ethernet card in a computer or a hard drive. This process is illustrated in Figure 7. The execution of plug and play design is not possible if manufacturers do not follow predefined standards.

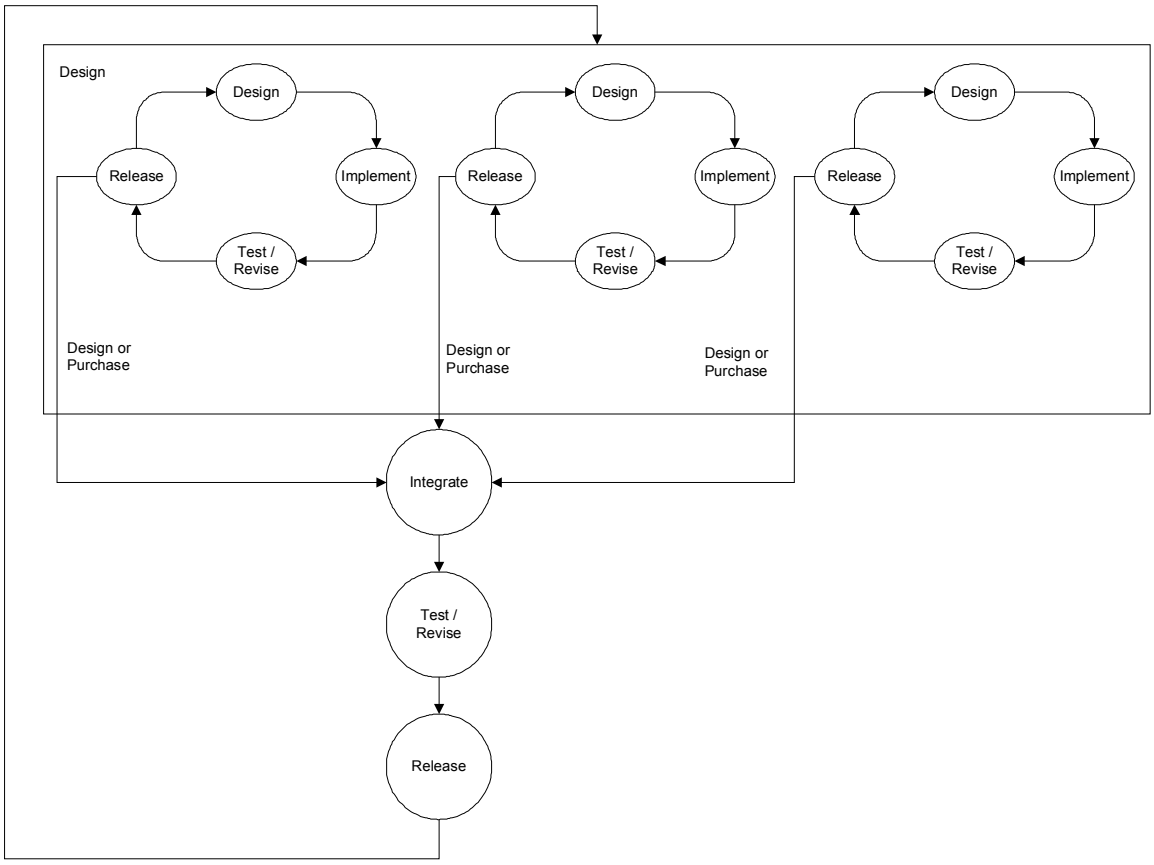


Figure 7: Integrated Plug and Play System Design

Ideally, the individual components should be able to undergo revisions independently. This introduces the definition of boundaries, and thus interfaces. Examples of these interfaces are:

- Functional
- Electrical
- Mechanical
- Thermal
- Human

More interfaces can be defined based on Figure 1. However, without standardized interfaces and behaviors, it would be impossible to have multiple components that can "plug" into the same spot.

One key difference to note here is that while the focus of this thesis is not to implement and design a complete plug and play power electronics system, it is to provide a communication and control platform that supports plug and play operation from a hardware perspective, as the PCI bus did for plug and play systems in the computer industry. With the addition of defined behaviors and partitions for specific module types, along with an enabling software architecture, a complete plug and play system will be specified.

2 System Architecture Overview and Multidisciplinary State of the Art

Partitioning and Distributions. Power converter subsystems can be classified on three axes as shown in [22]. Using these methodologies it is possible to partition systems into components. The three axes are temporal, spatial, and functional.

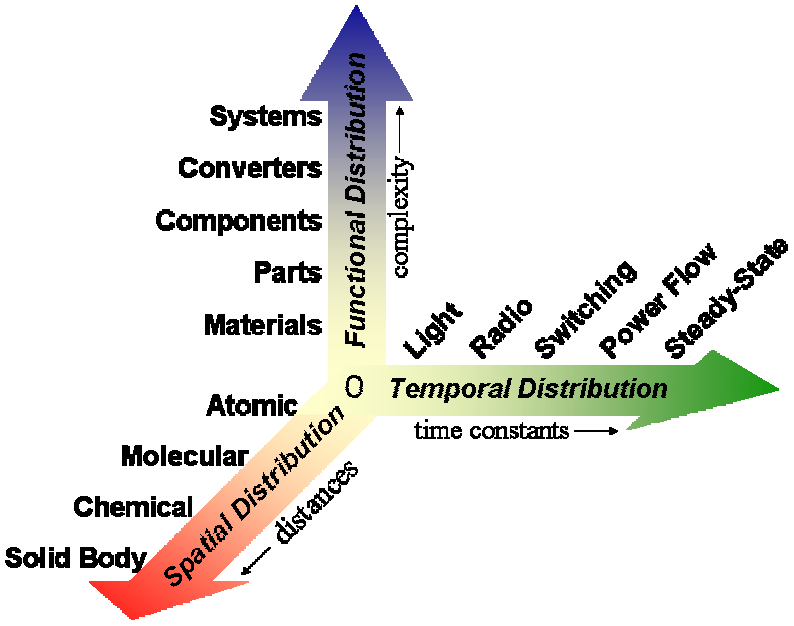


Figure 8: Dimensions of System Events and Information

The temporal axis describes events that happen as determined by the time constants involved, whether they are control, protection, or system dynamic functions. Some events are slow, such as changes in the reference, while some events are fast, such as switching action.

The spatial axis describes how things are related in terms of size and relative position. Some components and converter subsystems are spatially distributed, while others are small and compact. This is specifically of interest in high power converters.

The functional axis is a measure of how complex the subcomponents are. As components aggregate together to create complex applications, subsystems, and systems, the module moves up on this axis.

Whenever there is a partition and a separation of modules into atomic entities, there becomes a need to create interfaces that interconnect these modules together. Typically, these interfaces are customized.

These interfaces occur between components close together on these axes.

Hierarchical Partitioning. Converters can be partitioned using a hierarchical concept, as discussed in [23]. It is also proposed that the network and control architecture can be compared to the seven-layer open systems interconnect (OSI) model as shown in Figure 9. Examples are provided on the right hand at each associated level.

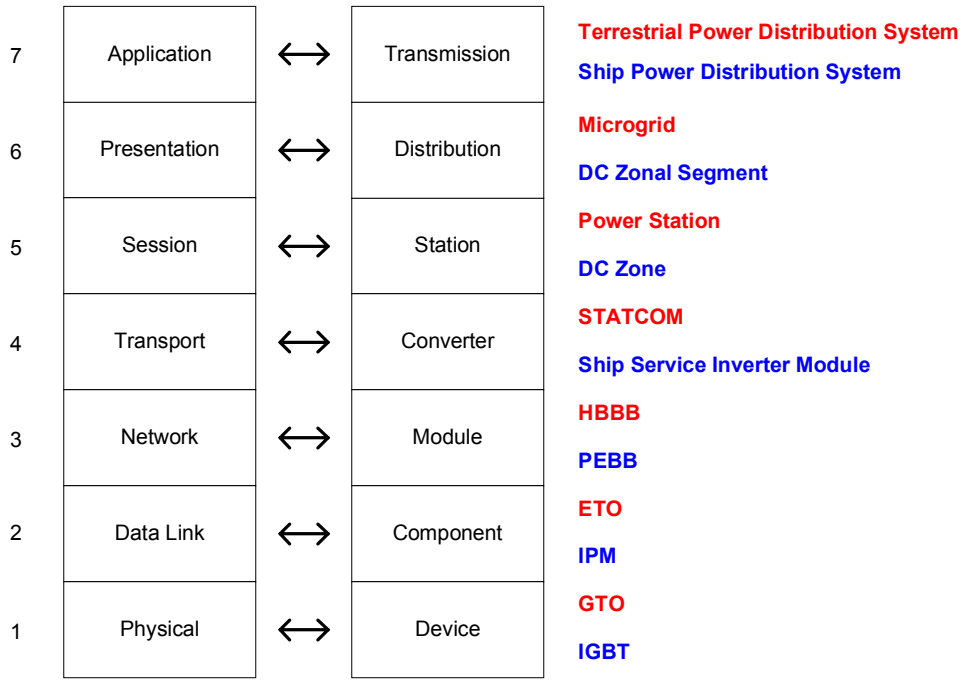


Figure 9: OSI Seven Layer Model Analogy

The architecture discussed here has been designed at the converter and module level (levels 3 and 4). The communication protocol proposed in this thesis is at the interface

between these two layers. It is proposed, however, that the protocol can be extended upwards.

This OSI layer model is an open model that allows the integration of devices from multiple vendors into a single application. The vendor independence has also been shown in applications such as PC personal computes, where different vendors focus on different strong points, giving rise to companies focusing on functions and modules such as an Ethernet card or a video card or a power supply.

This evolution takes place in the computer industry due to the introduction of standardized communication interfaces such as ISA, and was further advanced by plug and play with the introduction of the PCI bus [80]. Prior to that, computer systems were monolithic, and even if they were modular, they were not cross-compatible, and were sole-source (such as IBM or apple computer).

2.1 Plug and Play

Plug and play is a technique adopted by computer systems currently that allows systems to be constructed with minimal integration effort, allowing the primary focus of work to be application related. The actual hardware is not customized for the specific application, and instead, standardized modules are used to build an application.

The key characteristic to this type of system is the methodology used, and not any one specific implementation example.

Computer Systems Approach. For some time, computers have been monolithic, integrated devices. There was a single configuration of components. Interfaces were nonstandard, or proprietary. Replacement or expansion pieces had to be purchased for a specific computer model, as components were designed to work with specific host systems.

The introduction of standardized bus architectures, such as the ISA bus led to vendor independence. Different manufacturers could create equipment that had to be installed and configured in the converter. The standardization was in the hardware interface, and the device drivers and such still had to be written for the operating system and specific card.

The PCI bus [24] allowed for a plug and play implementation. The ability to configure devices from the PC operating system or BIOS directly led to increased flexibility and self-configuring systems. It allowed for the functionality of the module to be used at different levels. There was a device level, where the module provided a unique service that can be generally described. This allowed for drivers relating to those modules to be used without having to have a specific driver for every type of card. The drivers formed a hierarchy [25], and using plug and play, it is possible to control multiple vendor's cards using a simple generic driver. The functionality of the entire card may not be utilized, but the baseline functionality is available.

Currently with the addition of other bus types, the PCI bus becomes one option, as the same function may be implemented via the USB interface, and so there is a functional abstraction that allows the module to exist in several parts of the network, and it doesn't matter to the application how the module communicates with the computer.
Similar attempts have been made in weapons systems, as mentioned in [26].

Requirements of a Plug and Play System. In order to enable this, a set of characteristics for each module have been defined. Among these characteristics are the following:

- Module identification
- Self-configuration
- Reconfiguration and resource management
- Device classes, identification, generalized control
- Vendor independence / interchangeability

- Standardized interfaces,
- Standardized behavior

2.2 Proposed Power Conversion Systems Architecture Overview

High Power Systems Architecture Specification.

Power Converters consist of several functions:

- Control
- Sensing
- Actuation
- Communication
- Protection
- Thermal Management

These functions can be classified according to the three axes described before. Most power converters can be constructed modularly, defining system partitions on these three axes, grouping components in similar locations with similar functions together. These modules can be connected together via standardized.

It is possible to define a general control architecture to create modular components to meet application requirements, as done in [66].

The use of a standardized, modular architecture that is self-configuring, can be referred to as a plug and play architecture. The temporal and functional partitioning of the control along with plug and play give rise to a control hierarchy where supervisory modules coordinate the actions of multiple distributed modules, which operate in a synchronous manner.

Using a standardized communication bus, the supervisory controller can become hardware independent of the phase legs, abstracting the interface to the devices,

protection, control, local sensing, and such. It is desirable that this communication bus be flexible and survivable to expand the range of applications supported by this architecture.

The supervisory controller is referred to as a universal controller, or application manager in this architecture. It implements functions that are not specific to a specified phase leg or power module of the converter.

The slave modules being controlled are referred to as hardware managers. These hardware managers have functions of controlling power devices and sensing state variables. A hardware manager is also responsible for the local protection of the power devices and/or passives that it is associated with. The hardware manager may also implement local control functions, such as a tracking controller for a current, where the reference is provided by a universal controller or other supervisory module. This architecture is shown in Figure 10.

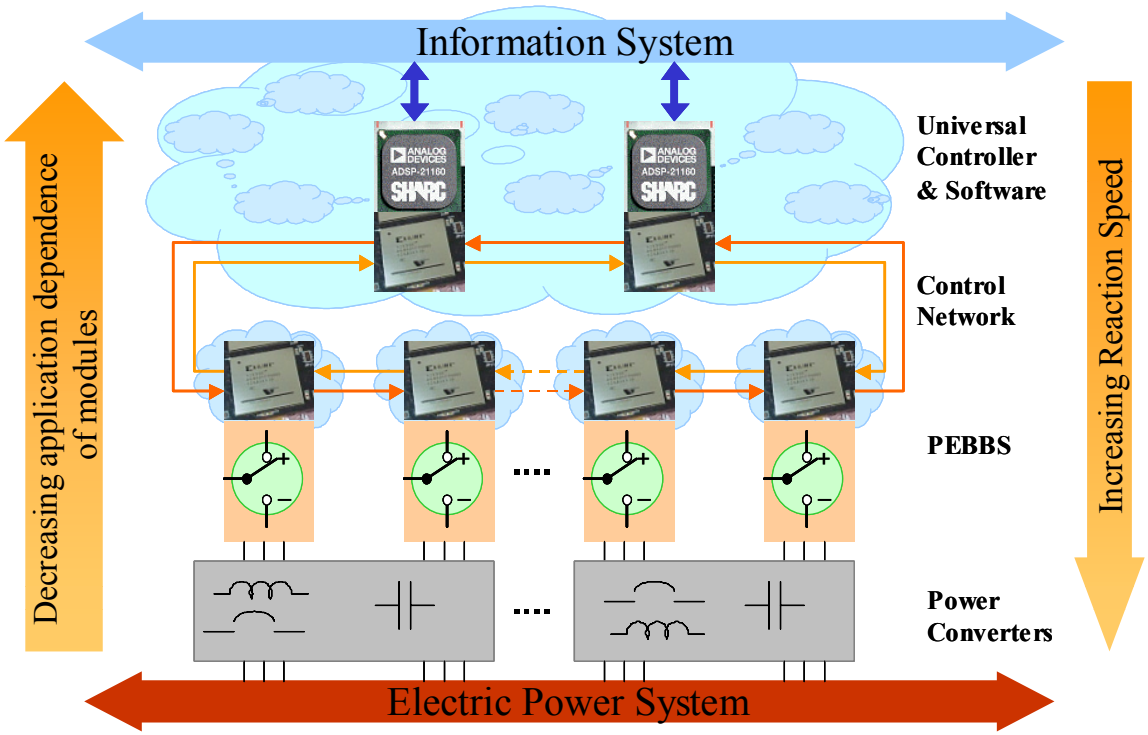


Figure 10: PEBB Plug and Play Systems Architecture

To connect the modules together into a working system, a synchronous control oriented communication protocol can be used. There is a communication between controllers and hardware managers. However, it is also possible for two controllers to communicate.

The control architecture should be able to handle a wide variety of power electronics applications varying from DC/DC converters to multilevel, multiphase and matrix topologies, requiring tight synchronization between switches.

The system examples used in this architecture focus on voltage source half-bridge topologies, although other topologies are also possible, such as cascaded H-bridge, matrix, DC/DC, current source, and SCR based converters.

The development of an architecture for digital control in power electronics is a multidisciplinary issue. Several areas contribute to this, including:

- Power Electronics
- Industrial Automation and field bus communications
- Digital Design and Computer Engineering

To provide background and motivation, some concepts from the state of the art in these fields that are used in this architecture design are presented here.

2.3 State of the Art in Industrial Automation

In many ways, the proposed digital control architecture is an extension of industrial automation control networks. Industrial automation networks have several features that can be used in a power electronics architecture:

Open Standard Field busses. Most industrial systems take advantage of one of the many available field busses to control their application. These field busses are open architecture, and have modules available from multiple vendors.

Distributed control. The control of processes in the plant are functions of separate programmable logic controllers (PLCs). The plant controls are not lumped into one

large controller that spans every device in the plant. This allows for modular operation and upgrades.

Decentralized I/O. The I/O of the system (both digital and analog) is not directly attached to a controller, but exists as a node on a control network. The controller will read the inputs and set the values of the outputs over this network.

Multimaster Systems. Some protocols have multimaster capability, allowing multiple masters to exist on the same network. These masters can sometimes communicate with each other, or control each other as if they were slaves on the network. Sometimes, a slave can have multiple masters.

Distributed visualization and monitoring tools. It is possible to monitor the status of the plant from anywhere within the network, or even over the internet. This service is not real-time, and is used normally for human operators to see the state of the manufacturing plant. Depending on the application, it may not be practical to monitor the status of a power converter over the internet unless it is a FACTS device or a substation. However, for applications such as an electric ship, it is desirable to know the status of the application.

Safe shutdown and definition of safe states. In the event of a failure or loss of communication, the devices will assume a safe state that does not endanger the operators and the equipment if possible.

Real-time control. Industrial automation plants utilize real-time control of their processes. This means that there is active feedback to a programmable logic controller that is using that feedback to determine the next actions that the plant should take. Typically, this reaction time is not faster than 1 millisecond.

Batch level control. The plant actions are coordinated at a high level, referred to as batch level control. This is a form of hierarchical control in manufacturing plants.

Data logging. It is possible to get the time history of the state variables within the plant for detection and diagnosis purposes, as well as for inventory control. Events are also stored in this database that are sent from the operator panels, or from the controllers.

Cell based manufacturing. The manufacturing plant is partitioned according to tasks. The same PLC that controls one manufacturing line may not be controlling another one. The manufacturing cells are functionally independent from other cells, and there is minimal coupling between them.

Figure 11 shows a sample cell in a plant. The top layer uses Ethernet, with visualization, data storage, and batch engines. This network typically runs on TCP/IP. These engines typically run on PCs (desktop, servers, or industrialized), and are not intended to be real-time controllers. The field busses connect components of higher speed real-time devices. The PLCs here are either hardware manufactured for that purpose, or PCs with software to make them real-time controllers [27, 28, 29]. It is important to notice the difference between the functions of the supervisory controllers and those of the PLCs. The supervisory controllers in this case manage control actions initiated by people or higher level plant functions. These actions have relatively long time constants. The commands may be similar to stop and start, perform an action, or a reference for a conveyor speed. That command, in turn, is sent to the PLC, which handles the intricacies of executing it. This essentially creates a hierarchy of control.

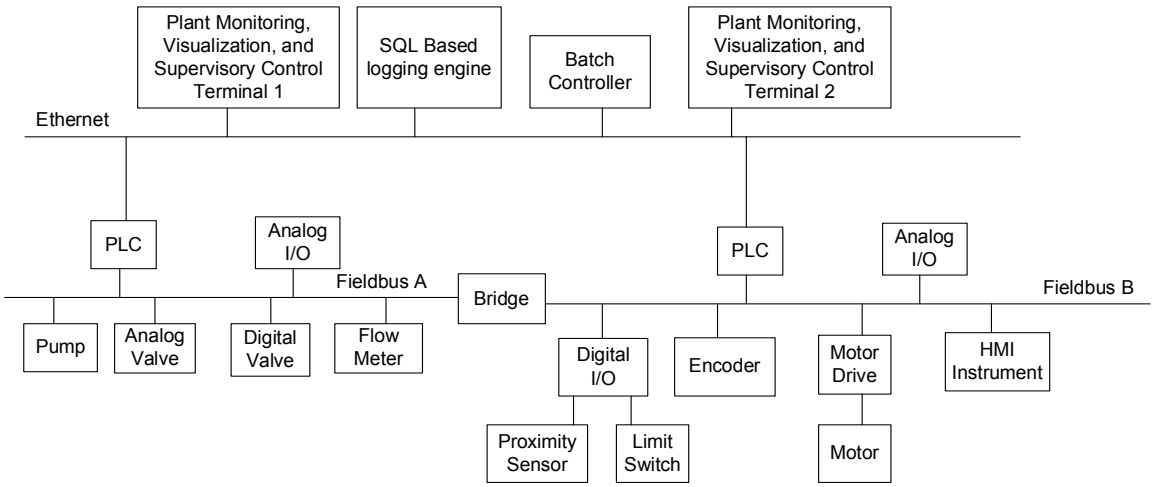


Figure 11: Sample Automation Cell

Industrial plants make use of field bus control networks. Several vendors and organizations have standardized these interfaces, and they are now an open standard. Examples of these are listed below:

- Profibus [30, 31, 32] (EN 50170, DIN 19245)
- DeviceNet [33]
- AS-i Bus [34]
- Industrial Ethernet
- Honeywell Smart Distributed System
- CanOpen
- ControlNet
- ModBus Plus
- LonWorks
- Interbus-S
- SERCOS [35]
- LightBus [36]

Each of these fieldbusses is controlled from an independent organization or consortium. The fieldbusses use open protocols that allows multiple vendors to produce products that can be connected to these busses. The controller(s) may be manufactured from a different vendor than the I/O modules.

Many of these protocols have several things in common:

- They ride on top of a lower level serial bus, such as RS-485, RS-422, or TCP/IP
- Most of these protocols, due to the lower level protocol, are nonroutable, but recently, many of them have been adapted with the ability to be encapsulated in a TCP packet.
- Most of these protocols execute at limited bandwidth (12 Mbaud Max)
- Most of these protocols allow for the addition of nodes on the network during operation, and can parameterize these devices during operation.
- Most of the I/O on these networks are dynamically configurable
- Therefore, most of these networks have a text file describing their operation, parameters, and capabilities (GSD file for Profibus, EDS file for DeviceNet)
- None of these fieldbusses support synchronization tight enough for use on a power electronics platform at the switch level.
- Many of these networks support multiple masters
- Many of these protocols can be converted to optic fiber

In addition, many of these protocols support different ways of communicating with other modules. For example, Profibus FMS allows master to master communication, along with redundancy, and multimaster control of a single slave module. It is also possible for modules in Profibus to be both a master and a slave to another master.

OSI Seven Layer Model. In networks, the International Standards Association has introduced the Open Systems Interconnection model [37], which is a seven layer hierarchical model that partitions control into functional layers which can be abstracted from each other. The seven layers are shown below.

Layer	Name	Description
7	Application	Application's Interface to Network
6	Presentation	Translates protocol data into application specific data.
5	Session	Establishes a logical connection between two nodes. Manages security.
4	Transport	Breaks up larger messages into smaller messages into smaller packets, and recomposes those messages from other nodes.
3	Network	Determines the route used to transmit the data.
2	Data Link	Ensures the correct reception of the network frames and error checking at the frame level
1	Phyiscal	Concerned with the physical transmission and reception of the message. Manages media used to transmit and receive signals.

Each layer presents information to and takes information from the layer above and below it. For example, in a serial protocol, as information comes in, it is translated from the physical serial stream to a frame of information that is forwarded to the next layer, the data link layer. The physical layer also takes frames of data and serializes them onto the physical media.

2.4 State of the Art in Power Electronics

Power electronics still uses centralized control for the plant, as shown before. For large systems, it is possible to separate these functions into structures similar to that of the manufacturing system, where the generic control functions implemented in PLCs become implemented in "Universal Controllers", and the power level and application specific components (the I/O, pumps, valves, and drives in the manufacturing analogy) will be implemented by hardware managers.

In order to further understand the relationship between control and the power stage, some fundamental topics in power electronics need to be introduced.

Feedback Control System. When a system is controlled, there are several options on how to control it. Typically, these systems have a mathematical reference model, which is typically a system of differential equations. Controllers can be used to regulate these systems to a set of performance objectives. Some systems use feedforward control, calculating the change in control variables necessary to produce

the desired output. The majority, however, use feedback control, which takes some measurements of the system to perform the control. Feedback control has several benefits [38]. They are listed below:

- Tracking reference signals with minimal steady state error
- Mitigating the effect of disturbances (Disturbance Rejection)
- Handling uncertainty and adding robustness
- Increasing plant Performance
- Stabilizing a system that may be unstable

Systems exist within environments that impose disturbances on them. Some of these disturbances occur as a side effect, and some are intentional, such as a change in reference. These disturbances are shown as d in Figure 12.

In a control system, there is a control objective – that is, what you want to achieve by implementing the control system. Deviations from the desired operating conditions can be considered as an error, shown as e in Figure 12.

A feedback controller must measure something (the feedback) from the plant in order to perform a control action. This measurement is shown as y in Figure 12.

A feedback controller must cause the system to change in some way. The controller's influence on the plant is shown as u in Figure 12.

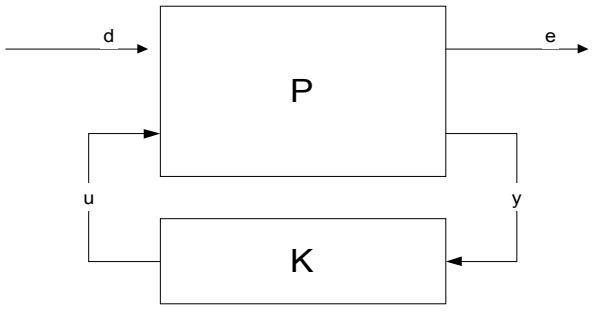


Figure 12: Feedback Control Block Diagram

There can be discrepancies between what is read as the measurement and the actual variable being measured that are introduced via the measurement technique and the environment. These undesirable effects are due to things such as noise on long cables measuring analog variables, uncertain sensor gains, or time-varying (due to heat or operating condition) sensor gains.

Three Phase System. In a three phase system, there are three voltages that are used to transfer power. Instead of the typical wall outlet, with a line, a neutral, and a ground, which is called single phase, there are typically three phases, a ground, and sometimes a neutral. The ideal three phase system will have voltages that appear like those in Figure 13.

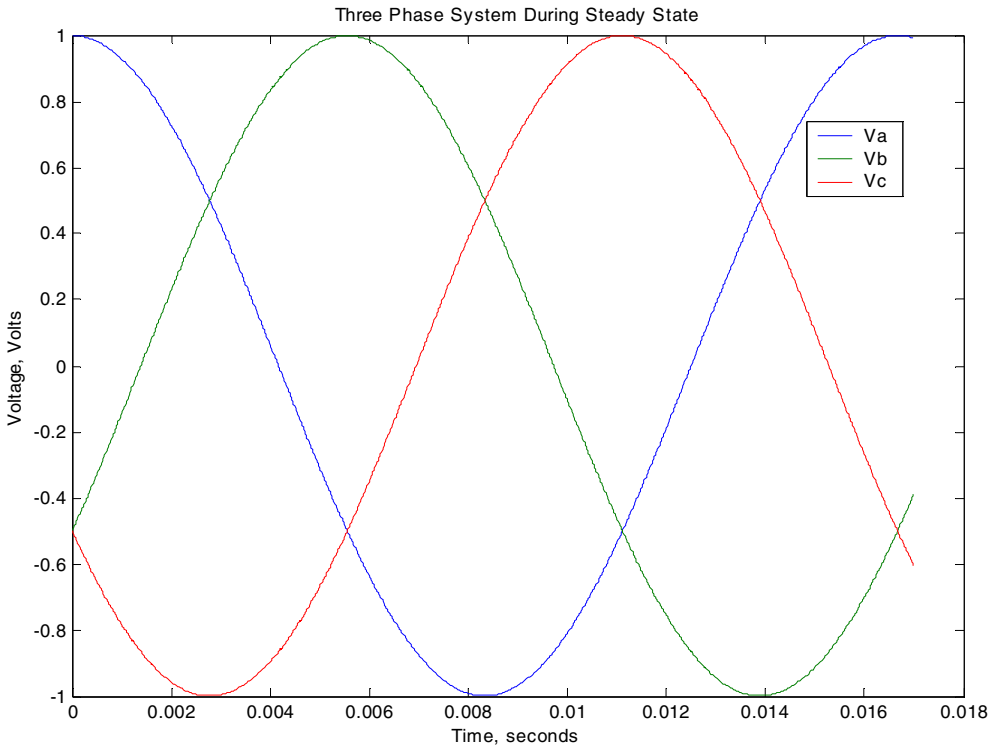


Figure 13: Three Phase System Variables

DQ Transformation. In a system with three phases, it is possible to represent those three phases as a vector variable in R^3, where each element in the vector is one of the

voltages of the phases. In a balanced system during steady state, the trajectory of the vector endpoint will "draw" a circle as the vector V rotates.

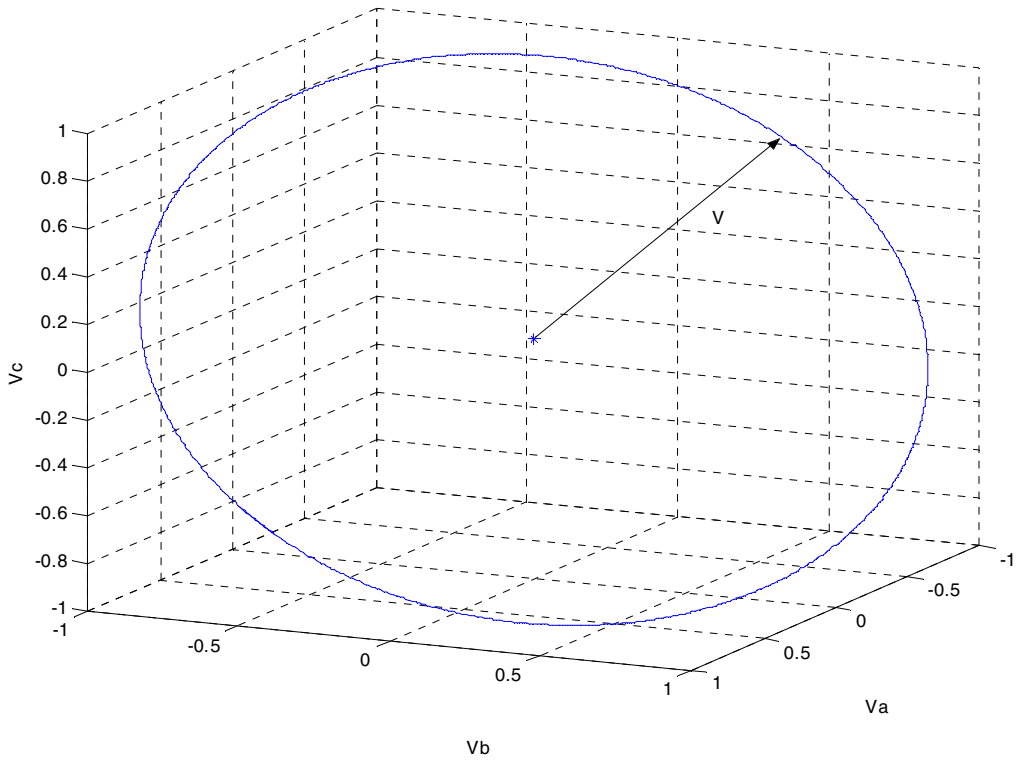


Figure 14: 3-Phase Steady State Trajectory in ABC Coordinates

It is then possible to define a new set of coordinate axes that have two normal axes on the plane defined by the circle, and the third one normal to the plane. The vector V can be represented under the new set of coordinates, and the elements of the transformed vector are called v_α, v_β, and v_γ. The new set of coordinates is referred to as $\alpha\beta\gamma$ coordinates. The γ component is zero here, as the vector rotates in the $\alpha\beta$ plane for a 3-wire system, and must obey the Kirchoff voltage and current laws. The trajectory also rotates with constant frequency at the line frequency. It is therefore possible to create a new set of coordinates by making the $\alpha\beta\gamma$ coordinate axes rotate (with a constant angular offset if desired). The coordinate axes will rotate about the γ axis. The new α axis will be called d, the new β axis will be called q, and the γ axis will remain as it is, but it will be called 0. Therefore, the DQ0 coordinates are the rotating version of the $\alpha\beta\gamma$ coordinates. Since everything is rotating at the same

frequency in the same direction, the relative position of V to the DQ0 axes becomes a constant, turning a once dynamic system into a static system. Doing this creates an operating point that allows the system to be controlled as if it was a DC-DC converter [39]. In order to perform this transformation, it is important to remember that knowledge of at least two components of V are needed if the system is 3 phase, 3-wire, and all three are needed otherwise, and therefore this transform must be implemented at a higher level than the individual phase legs of the system. In the architecture presented here, it may be implemented in a universal controller.

Pulse Width Modulation. Power electronics circuits work by turning power semiconductors on and off. Operation in the active region is undesirable. To understand the operation of a power electronics circuit, a simple example is given below:

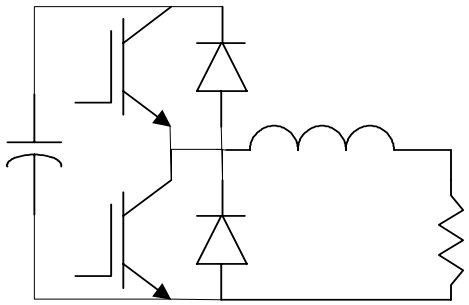


Figure 15: Half Bridge Structure

If the voltage across the capacitor (referred to as V_{DC}) is 800 volts, and 600 volts was desired at the output of the resistor, it is possible to define a time called a switching cycle, and turn the top switch on for half of the time, and then turn it off, and turn the bottom switch on for half of the time, and then turn it off. Repeating this continually at a high frequency, the average voltage at the midpoint of the two switches will be 600 volts. The ratio of the time that the top switch is turned on to the total switching period is called the duty cycle. The inductor provides a filter to attenuate the high frequency component caused by the switching of the two devices, as shown in Figure 16. This technique is called pulse width modulation. This is shown here for a single output voltage. More information can be found in [40].

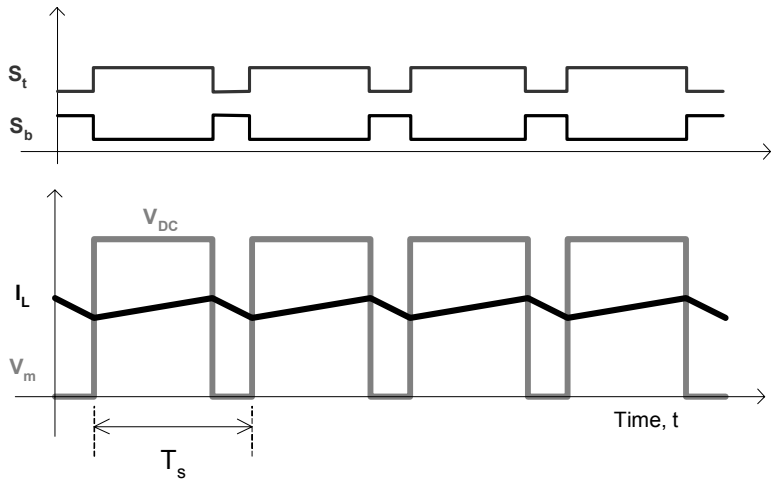


Figure 16: Pulse-Width Modulation Waveform

When the switch is on, the current will flow from the DC link to the load via the inductor, as shown in Figure 17.

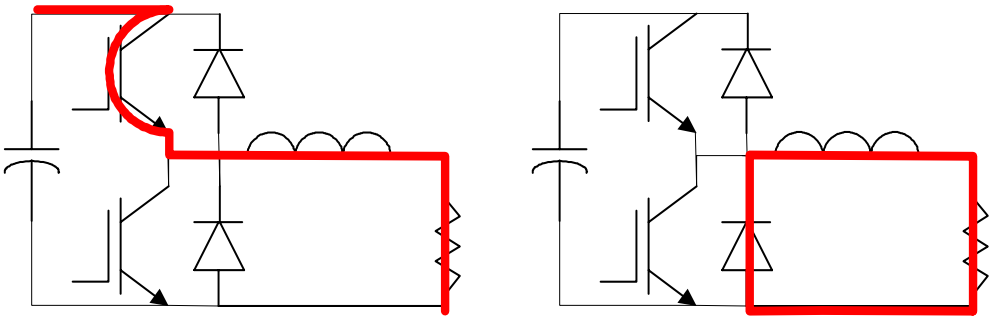


Figure 17: Current Paths: Left (St = '1', $I_L > 0$); Right (S_t = '0', $I_L > 0$)

Three Phase Converter System. To use power electronics in three phase systems, the concept of a three phase converter is used. A three phase converter is an aggregation of single phase components that are controlled in a new way to produce the desired trajectory of three phase variables. An example of a three phase – four wire voltage source converter driving an inductive load is given in Figure 18.

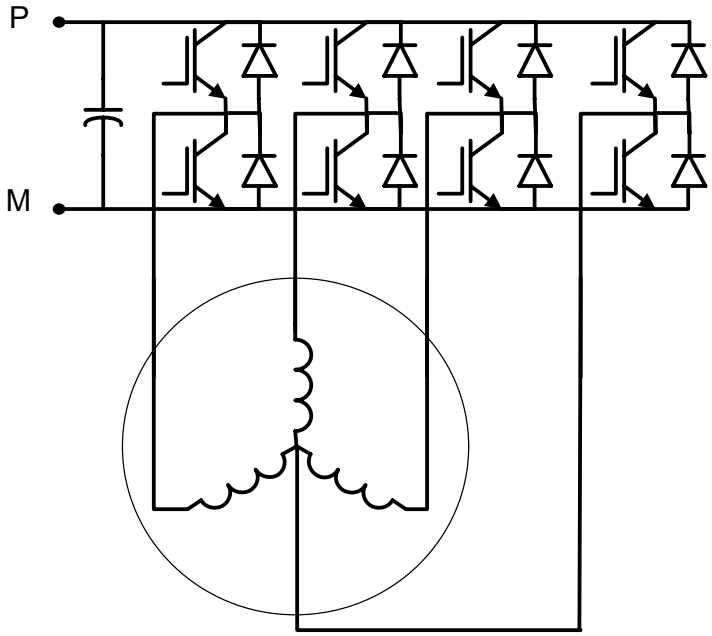


Figure 18: Three-Phase 4-Wire Converter

Converters can be placed back to back to make more advanced converters, such as a frequency converter, which is a combination of a rectifier and an inverter state, as shown in Figure 19. A power converter can have a large number of phase legs (and therefore switches) operating together. This example has 14 switches.

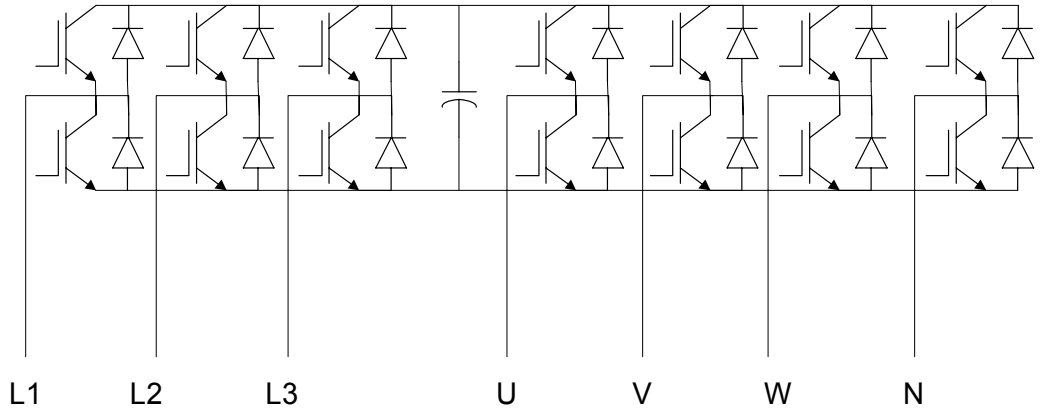


Figure 19: Frequency Converter (AC-DC-AC)

Space Vector Modulation. In a three-phase system, there are three of these voltages, and therefore, three of these structures. At any given instant in time, for each half-bridge, one switch is off, and one switch is on. It is possible to represent the voltage

between two of these switches in αβγ coordinates. Those voltages will define a vector for that instant. There are eight possible vectors if we assume we are on the αβ plane, including two vectors in which the voltage is zero. It is possible to construct a vector in space by quickly switching between two or more vectors defined by these switch combinations, referred to as switching vectors. The sum of the time that these three vectors are active is equal to the switching period. This concept is shown in Figure 20. The technique can be directly related to pulse width modulation, but instead of modulating a single scalar voltage, here the concept is to modulate vectors.

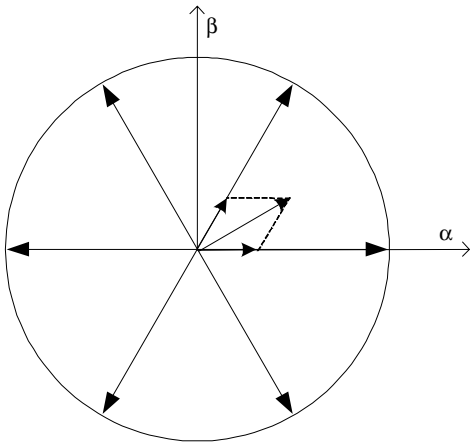


Figure 20: Switching Vectors

It is then possible to turn the resulting vectors back into phase duty cycles that can be used to control each half-bridge pair to effectively get the resulting vector in the real system. This vector rotates around the gamma axis to produce the same waveforms as shown in Figure 13.

In order to perform most of vector operations, knowledge of variables relating to more than one phase is necessary, meaning that this function must be implemented at a point in the converter where information from all phases is available. This control is therefore a converter level function, and not a phase leg level function.

Once per switching cycle, all of the state variables are read, converted to DQ coordinates, a control law executes, the variables are then brought back to

αβ coordinates, where they are decomposed into switching vectors. The switching vectors are then decomposed into duty cycles, which are inputs to the pulse width modulators for each phase. An example for sector 1 is given in Figure 21, where V_1 (PNN) is active for time T_1, and V_2 (PPN) is active for T_2. The zero vector was chosen to be PPP. This example uses SVM4 as described in [41].

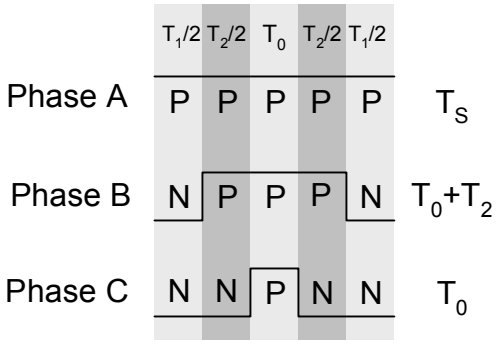


Figure 21: Duty Cycle Computation from Space Vectors

Multilevel Topologies. When a single device does not have enough blocking capability to block the entire voltage, two or more switches are needed to block the voltage. An additional benefit to having more switches is that it is possible to create multiple voltage (or current) levels, which may reduce harmonics caused by switching when controlled properly [42 ,43]. The former half-bridge can be turned into a multilevel half-bridge using diodes as shown in Figure 22.

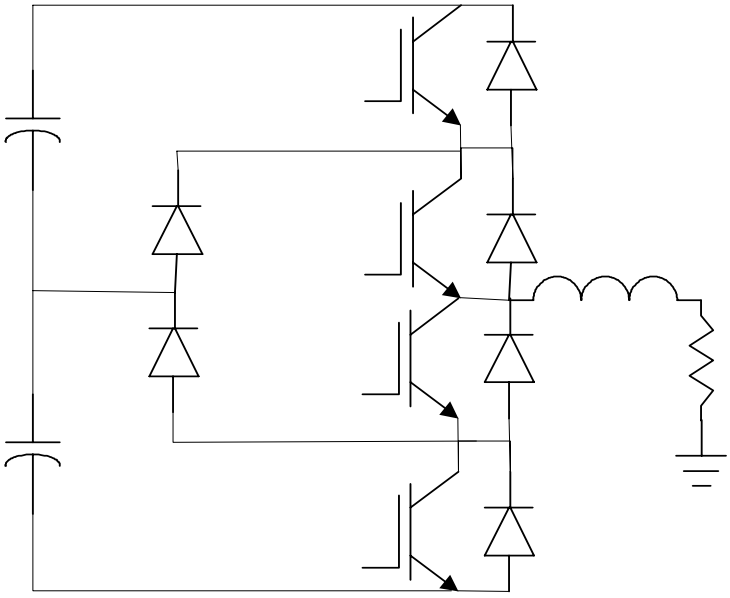


Figure 22: Multilevel Neutral Point Clamped Half-Bridge

The three levels in the bridge can be achieved using the following three combinations of switches shown in Figure 23. The left one produces the DC bus voltage on the output, the middle one produces the voltage on the top dc rail, the middle one produces the voltage at the midpoint, which is halfway between the positive and negative dc rails ideally, and the bottom one produces the negative dc rail voltage at the output.

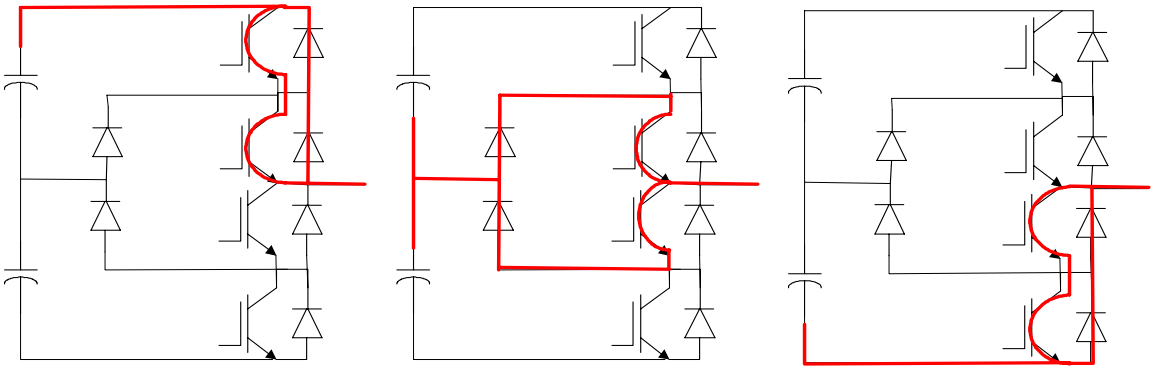


Figure 23: Switch Combinations in 3-Level NPC Half-Bridge

When switching between any of these states, if less than two switches are blocking voltage, the entire DC bus will be blocked across one switch, causing it to fail. This multilevel PEBB can be redrawn using the switch concept before as shown in Figure 24 [44]. Notice that in order to realize this concept, all four switches must be controlled to act as one switch.

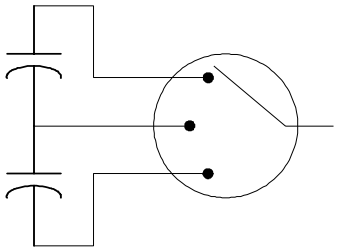


Figure 24: SPTT Switch

If a multilevel converter was used to create a frequency changer, such as the one shown in Figure 19, the resulting topology would have 28 switches for a three level converter, or 56 switches for a five level converter, along with a complex modulation scheme. A five level converter receives a fault status signal from each switch, as well as sends a command to each switch, which would produce a total of 112 control signals going to and from the gate drivers alone. Additional auxiliary switches used in soft switching would add even more switches to the configuration. As it can be seen for higher power converters, especially multilevel ones, the complexity, and number of control signals grows at a very fast rate, and a centralized control scheme would be very hard to justify from a control, development, and maintenance perspective, especially for reconfigurable applications.

2.5 State of the Art in Digital Design and Computer Engineering

In digital design and embedded control, there are multiple ways to implement controllers. One uses sequential logic, and the other uses concurrent logic.

Sequential Logic. Sequential logic is more naturally aligned with human thought. An example would be an instruction manual. First, do something, then second, do something else, and the list goes on like that. This is typically the type of language used when programming code for microprocessors. The code exists as a series of instructions with some sort of flow control. Sequential logic is typically implemented in a processor such as a DSP, or digital signal processor. Sequential logic can be represented by a flow chart.

Multithreaded Sequential Logic. Multithreading is the concept of having several streams of sequential logic (referred to as threads) executing at the same time. There are implicit challenges in having multithreaded systems. The most important is synchronization and resource sharing [45, 46].

Concurrent Logic. Concurrent logic, on the other hand, is less intuitive. It can be described as a set of things that happen at the same time. One subset of these types of processes is synchronous logic, which is driven by a clock. Data is valid at an edge of the clock, and is processed between these edges. A good example of concurrent logic is a pipelined multiplier. The structure of the multiplier is shown in Figure 25. Here, a two digit number a, composed of a_1a_2 is being multiplied by a one digit number, b_0. The process uses only single digit adders. The shift left operation does not take any time, and is just a matter of selecting the different address lines. Here, during any given clock cycle, a1 is multiplied by b_0, a_0 is multiplied by b_0, and the sum of the previous multiplication is added to produce an output.

Concurrent logic can be described using VHDL [47, 48] (VHSIC Hardware Description Language) code. This can be described by the following code:

```
process( CLK ) is
begin
      if( rising_edge(CLK) ) then
            FF1 <= a1 * b0;
            FF2 <= a0 * b0;
      end if;
end process;

y <= FF1 & "0000" + FF2;
```

In synchronous logic, the factor limiting the frequency of the clock is the longest propagation delay between any two flip flops. A flip flop contains a state variable. Complex logic structures that have many complex operations going on in the same clock

cycle cause the maximum clock speed of the entire system to drop, even unrelated tasks controlled from the same clock.

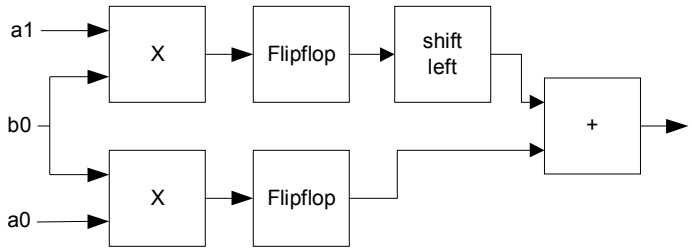


Figure 25: Pipelined Multiplier Structure

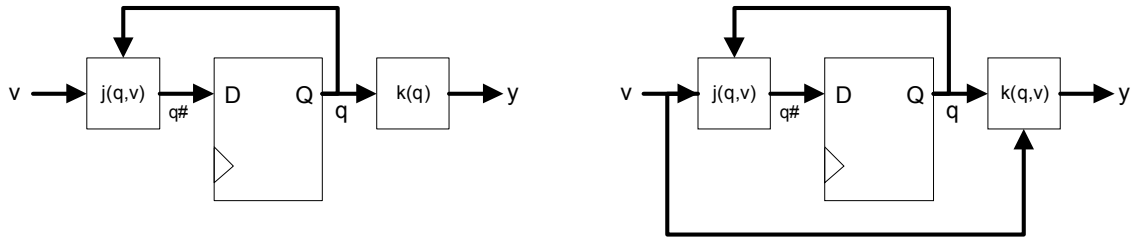
Pipelining. Pipelining is the process of taking a task that is complicated or time consuming, breaking it down into stages, and executing those stages in parallel on a stream of data [49]. It is possible to send data through the pipeline, and produce results at faster rates than performing the task on a piece of data one by one. In the example above, during clock cycle one, two numbers can be submitted at the input of the multipliers. The next clock cycle, two more numbers can be submitted. At this clock cycle, the first two will have been added, and are at the output, and the third clock cycle, the next data set will be ready. This allows the apparent speed of the multiplier to double when processing large quantities of data. There is an initial delay when the first piece of data propagates through the pipeline, and then there is an end delay until the last piece of data leaves the pipeline. Pipelining is a technique that is used to process sequential logic, as well as to process concurrent logic. An example of the pipeline is given in Figure 25.

CLOCK	1	2	3	4	5	6
a1	1	4	1	0	2	
a0	2	5	2	9	1	
b1	3	2	6	7	4	
ff1		3	8	6	0	8
ff2		6	10	12	63	4
y		36	90	72	63	84

Figure 26: Time History of Pipelined Multiplier Process

This example shows how pipelining can accelerate the process of multiplying two numbers. During clock cycle 1, two numbers (12 and 3) are multiplied together. The two digits of 12 are individually multiplied together, and placed at the output of ff1 and ff2. They are then added together (with appropriate shifting) to produce the output, y. It takes two cycles to multiply two numbers together. However, due to pipelining, it only takes 6 clock cycles to multiply 5 sets of numbers.

Synchronous Finite State Machines. A synchronous finite state machine (FSM) [50] can be described using a logic function that is a function of the current states and current inputs. Synchronous state machines are driven by a clock. There is a set of possible states that the state machine can take, and a set of rules that determines the next state based on the current state and current inputs. There is an output function that describes the outputs of the state machine. Synchronous FSMs can be divided into two types. A Mealey state machine and a Moore state machine. In a Moore state machine, the output is only a function of the current states. In a Mealey state machine, the output is a function of both the current states as well as the current inputs. The output of a Mealey state machine is therefore asynchronous with the clock, as the inputs can change between clocks, which would change the value of the output. However, the state transitions are still functions of the clock, and so the Mealey state machine is still considered synchrounous. Diagrams of Mealy and Moore state machines are shown in Figure 27.



$q^{\#} = j(q,v)$
$y = k(q)$

$q^{\#} = j(q,v)$
$y = k(q,v)$

Figure 27: Moore and Mealey State Machines

Field Programmable Gate Arrays. An FPGA is a reconfigurable logic device. While a DSP is a device that executes sequential logic, an FPGA is a device that "executes" concurrent logic. An FPGA is composed of many combinational logic blocks, or CLBs. Each CLB has a lookup table, and two flip-flops. As FPGAs become more advanced, they acquire more capabilities, more CLBs, and become faster. An FPGA is coded using a hardware description language such as VHDL or Verilog. Between the CLBs of the FPGA are a set of programmable interconnects to configure the flow data. The data in the lookup table in the FPGAs, as well as the settings for the interconnects are configured using an underlying layer of SRAM. The SRAM is loaded when the FPGA is configured, and this defines the behavior that the FPGA will have. On the outside ring of the FPGA, there are IO blocks, or IOBs. These IOBs may be configured to support a variety of different logic families. The IOBs also can tristate the data signal for signals connected to a shared bus, meaning that the output appears as very high impedance, allowing other devices to drive the same line that the FPGA is sharing with it. FPGAs are becoming more advanced. Today, Xilinx makes FPGAs with up to four PowerPCs embedded in them, allowing sequential logic in a concurrent environment [51].

One of the primary benefits of FPGAs is their ability to reconfigure the code that is executing inside them (either complete or partial reconfiguration). Some of them even support runtime reconfiguration, but all of them are at least programmable prior to runtime. Much work has been done on runtime reconfigurable FPGAs [52].

3 Universal Controller

A universal controller is responsible for all control functions that are supervisory in nature to any power electronics building block at level 3 in the hierarchy described in Figure 9. It contains the control algorithm for the converter level of operations, and in doing so, may collaborate with other controllers at this level, as discussed in [53].
Uses of Controller:

- Distributed Control
- Protocol Analysis / Converter debugging
- Phase Leg Emulation
- System Monitoring
- Upper level supervision of other controllers

3.1 Specifications

There are several features desirable in a distributed controller. The most obvious is the need for a CPU to execute code written to control the application. The controller should be able to communicate with the external world in which it exists. This is done via standardized system interfaces. The standardized interfaces allow the controller to be quickly integrated into a system without having to design customized interfaces and support. Scalability is also a desirable property. Applications will require more than one controller. There should be a way for multiple controllers to work together to solve large problems. It is impossible to predict and implement every requirement for every application. There should be some way to expand the controller capability for unpredictable future needs. A block diagram of the fundamental controller properties is shown in Figure 28.

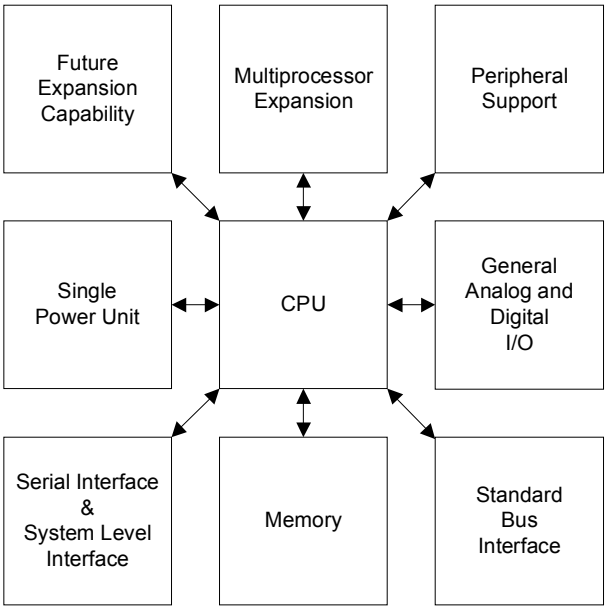


Figure 28: Controller Functional Block Diagram

It is desirable to have a modular approach to implementation so that each module can be developed and tested incrementally. Once one module is debugged, it should remain functional while other modules are being developed. It should also be possible to deactivate one module if it is not required in the design. Being universal means that the controller should be portable across applications. The strategy to implement the controller should support reconfiguration to allow this.

3.2 Approach

3.2.1 Architecture

The controller architecture consists of two main busses bridged by the FPGA. This approach allows for incremental debugging of the controller without having to worry about every block at once. This also allows for future architectures in which the FPGA could interact with the peripherals without having to know what the DSP is doing. An alternative would be to have a singe bus with every peripheral connected to it. The controller architecture is shown in Figure 29.

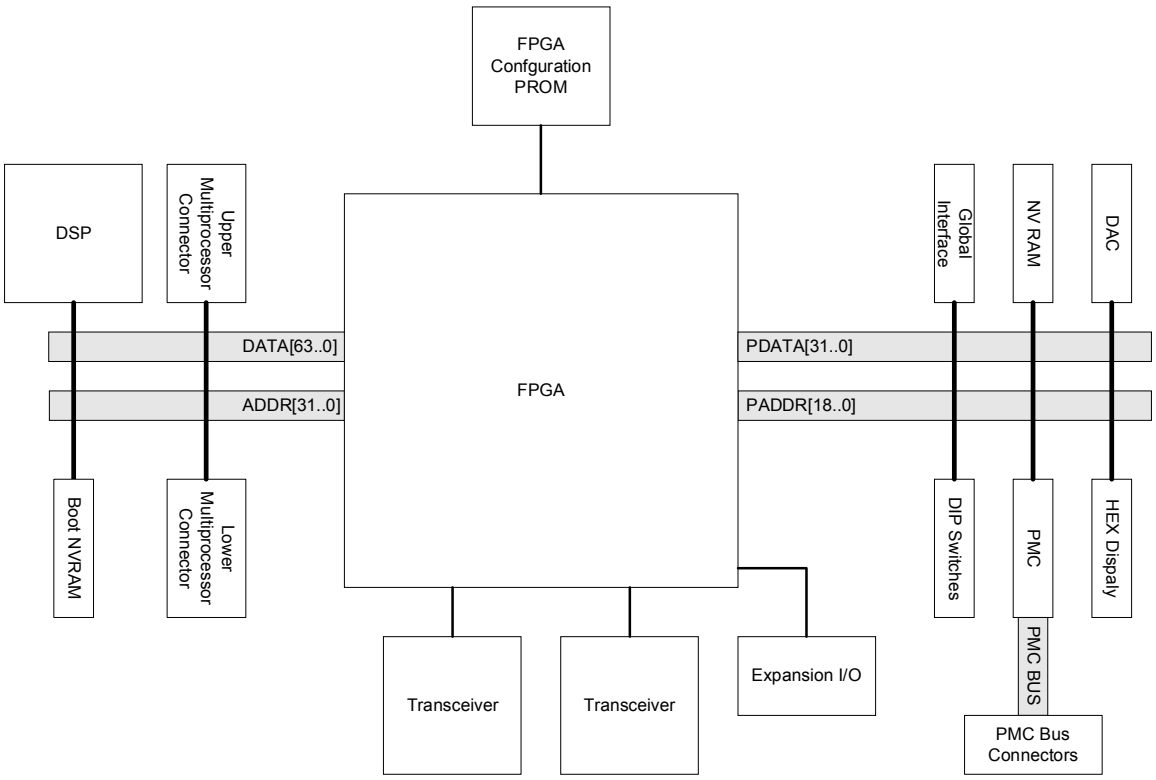


Figure 29: Universal Controller Architecture

There were two approaches possible when implementing the FPGA control code. The first approach allowed the thread of execution to pass from the DSP, through the FPGA, and into to the ASIC, which would service the request, return control to the FPGA, which would in turn return control to the DSP.

The second approach is a multithreaded approach, where the DSP would interface with a set of control registers in the FPGA that would perform a specific task, most likely involving an ASIC. This architecture would require the FPGA to implement a command processor that would execute the commands placed in the command buffer in the FPGA by the DSP, thus creating a second thread of execution. The second thread of execution would need to synchronize with the first one in some way. This complicates the interface between the DSP and the FPGA. It would also mean that after the write to the FPGA

returns, there would be no direct way to tell if the operation has completed or not without either polling a register or using some interrupt and callback function []. Reads from ASICs would also be complicated, as the data is not immediately available in the FPGA, and so there would first have to be a request, and then later, the data would become available after the FPGA had retrieved it.

These two approaches are illustrated in Figure 30.

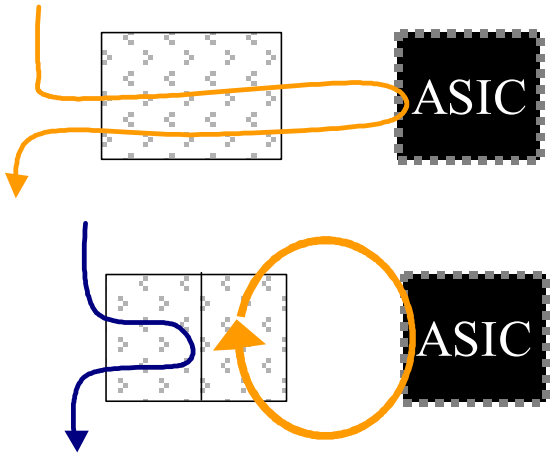


Figure 30: FPGA Control Strategies

After choosing the first architecture, it was necessary to choose a method to implement it. Originally, each feature of the controller had its own control block, and when the DSP requested that function, a control block assigned to that ASIC would take control, manage data and addresses, and locally store data. This architecture is shown in Figure 31.

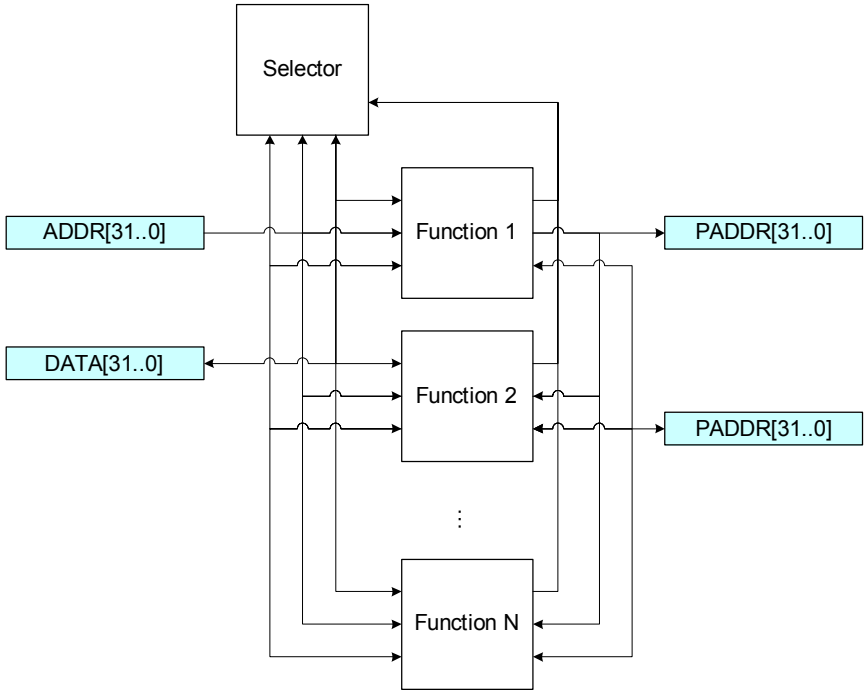


Figure 31: Original FPGA Architecture

As more and more control blocks were implemented, the maximum clock speed of the FPGA started to drop very fast. Eventually, it became difficult to manage each block due to the increased propagation delay within the FPGA. The propagation delay may not be necessarily through a CLB, but may be through routing logic, especially if the same signal is used in many places. Each control block would have to have address, data, and control lines. Most blocks passed data directly from the DSP to the FPGA directly instead of manipulating the data. Taking this into account, a new architecture was developed that allowed the control blocks to control the data flow while not actually "seeing" the data themselves. Control data that is stored in the FPGA to set some control line, such as the blanking control for the hex display goes to a set of registers. This new approach is shown in Figure 32.

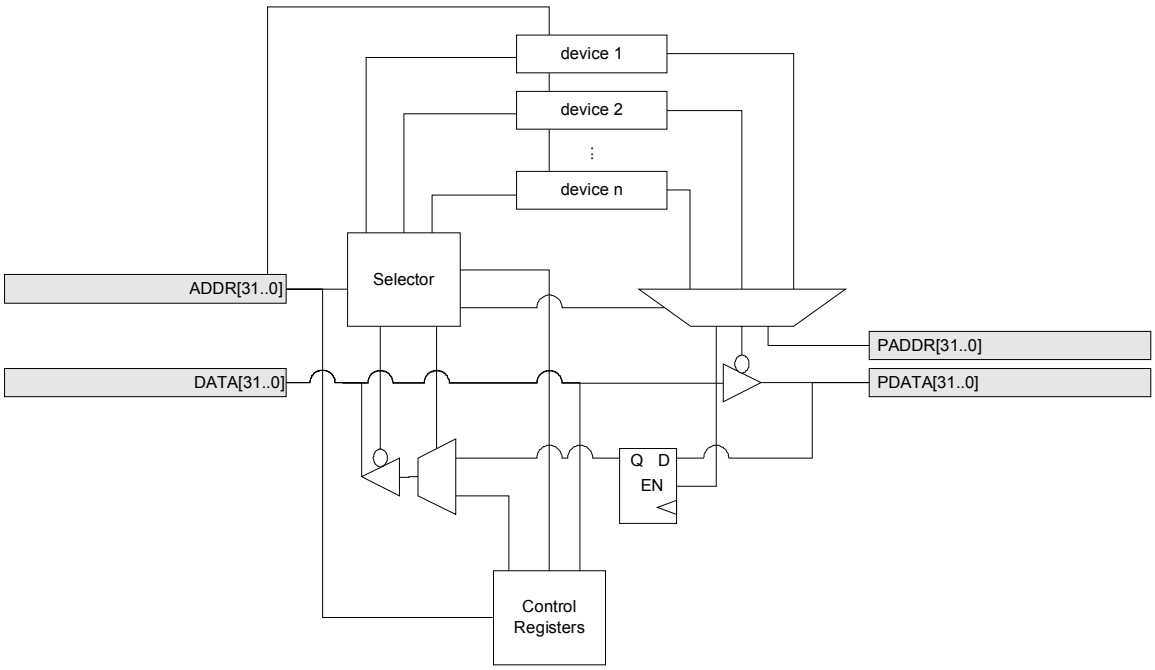


Figure 32: Improved Bus Management Based Architecture

In this architecture, several controllers can be connected together for tasks that require larger processing power. When the controllers are stacked, the DSPs and FPGAs for all boards appear in parallel with each other as shown in Figure 33. The FPGA uses the ID of the DSP as part of its address scheme. It is therefore possible for a DSP to access a resource on the FPGA of another board as easily as it would access a resource on one of its own.

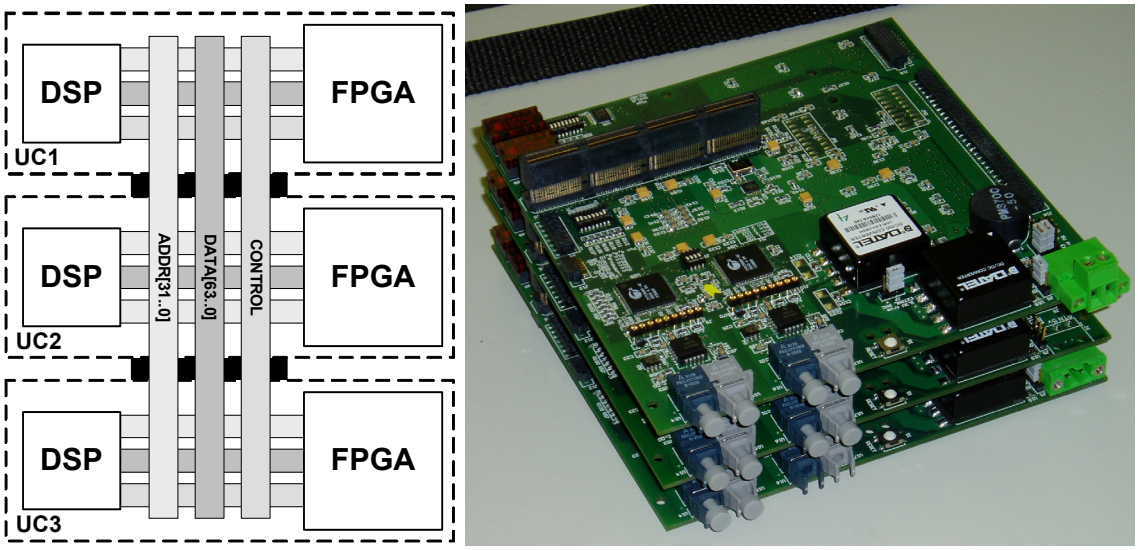


Figure 33: Stacked Controllers: Left (Schematic), Right (Physical)

3.2.2 Block Descriptions

The controller was constructed based on these requirements, and a set of components were chosen. The FPGA bridged the two busses, and provided a memory map that allowed the components to appear as memory locations to the DSP. Figure 34 and Figure 35 show the controller after it was manufactured. This section describes the controller features, and how they were integrated into the system for access via the DSP.

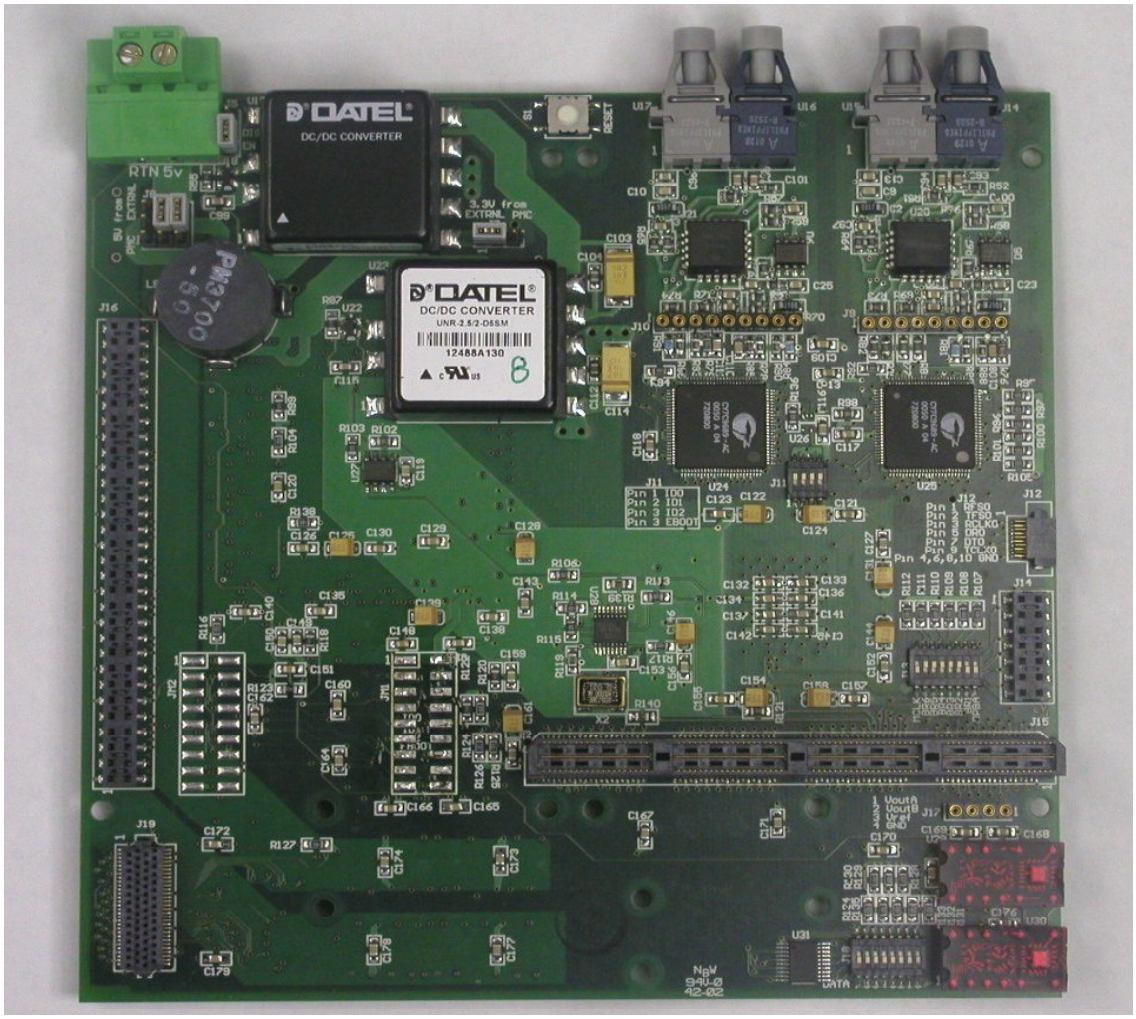


Figure 34: Universal Controller Front

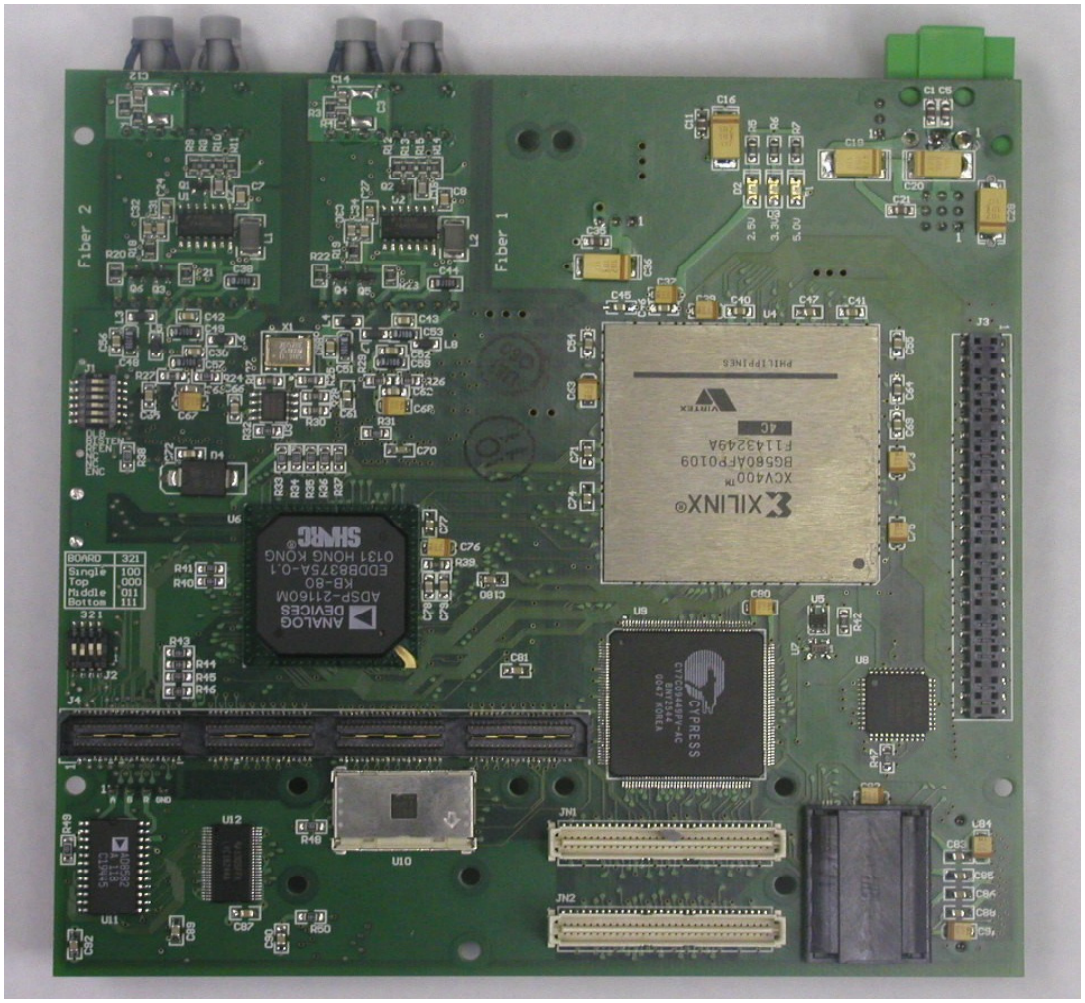


Figure 35: Universal Controller Back

3.2.2.1 DSP

The DSP chosen is an Analog Devices ADSP-21160 80 MHz floating point DSP. The DSP has the following features that make it attractive for use in advanced control architectures:

- Single Instruction, Multiple Data execution
- Intrinsic support for multi-processing
- Built-in multiply and accumulator, barrel shifter, and ALU
- Pipelined execution and instruction loading via Data and Address Generators
- Host Port Interface
- 2 Synchronous Serial Ports

- Built-in control of external port

The FPGA supports the DSP's access to other peripherals on the controller. The DSP interface has been designed according to the architecture described in [63]. The DSP communicates with the FPGA using the data bus, the address bus, and some control signals as shown in Figure 36.

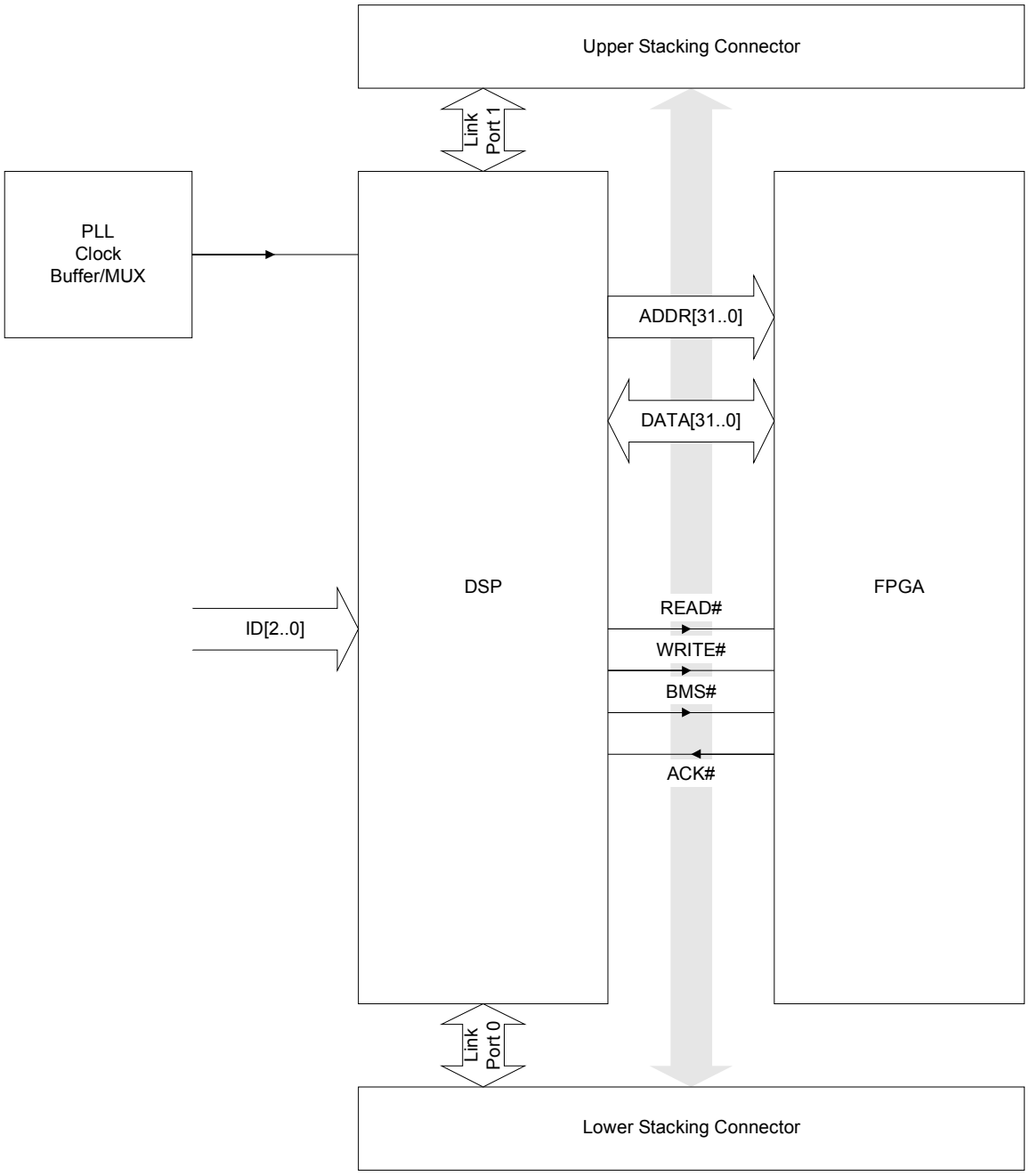


Figure 36: DSP-FPGA Interface

When the DSP sends information to the FPGA, it first sets up the data and addresses, and then controls the READ# and WRITE# signals. The DSP supports different types of peripherals intrinsically such as SDRAM and SBSRAM. To simplify the interface to the DSP, the FPGA interface was chosen to emulate SRAM. While the FPGA was processing the DSP's request, the FPGA would hold the DSP in a wait state until it is finished. This is shown in the following timing diagram illustrating an interaction between the DSP and the FPGA.

The DSP is packaged in a 400-ball Ball Grid Array (BGA) package. Some advantages to this packaging include the increased heat transfer due to the large concentration of vias to power planes, which become heat sinks to distribute the heat introduced by the processor. Another advantage is the small footprint that the device takes up, allowing for a higher density of components. However, this high density introduces some challenges. There is an increased difficulty associated with routing traces in the PCB coming in to and going out from the BGA due to the large number of vias obstructing traces, even on lower layers. Another challenge that increased the cost was manufacturing. BGA components introduce the challenge of mounting that is not encountered with other packages. The PCB has to be relatively flat to mount the BGA, and copper imbalance and other artwork and manufacturing details may cause the PCB to warp and twist, which must be avoided to successfully mount this device. Associated with mounting are the process requirements necessary in order to place the BGA on the PCB. There is a challenging process that must be followed in order to successfully reflow the device and then ensure that it is connected. An X-Ray is taken of the PCB to ensure that all of the solder balls have been successfully connected to the associated pads. A final challenge comes in debugging. There is no access to the pins of the BGA component. Debugging methods cannot rely on the values of the signals at the pins of the device, as they are not available for measurement. Therefore, it may be desirable to consider debugging when designing the PCB.

The FPGA is selected when ADDR[31..29] is equal to the processor ID as defined by the DSP-ID DIP switches, and when either one of the write signals or one of the read signals

are low. Due to the propagation delay within the FPGA, an extra clock cycle was added to the SELECTED signal to allow every input signal to stabilize before using those signals to decide on the next state or latch data.

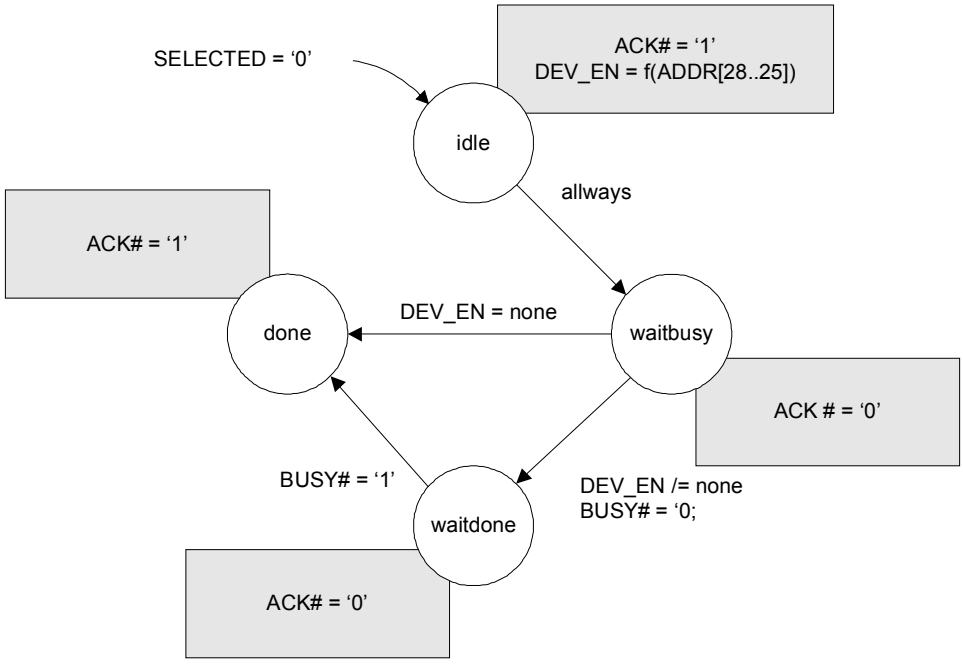


Figure 37: Selector State Machine

3.2.2.2 FPGA

On start-up, the FPGA is configured from a Xilinx configuration flash, XC18V04-VQ44C [54]. There are several configuration modes supported by the flash and the FPGA. The one chosen was the parallel SelectMap [55] configuration method. In this method, data is loaded into the FPGA one byte per clock cycle. SelectMap also allows other features for advanced debugging and configuration that can be used via the JM1 and JM2 connectors and the Multilinx programming cable. Due to time constraints, the JTAG interface of the Multilinx connector was used, although the entire interface is still available.

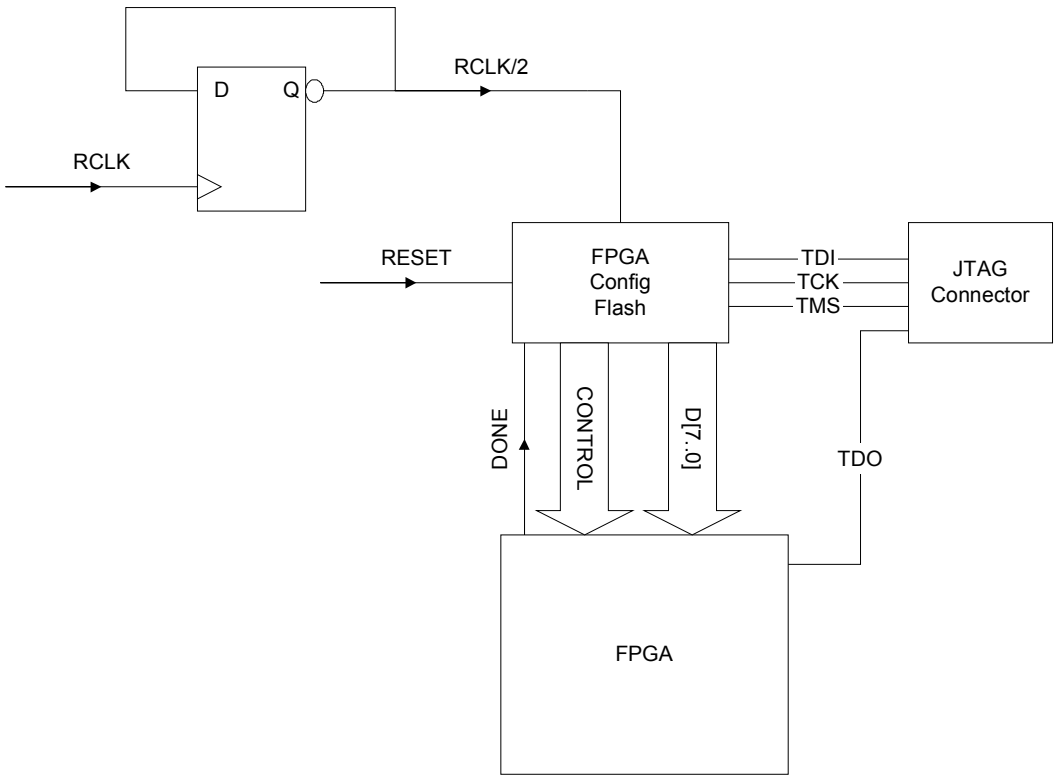


Figure 38: FPGA Configuration Circuit

3.2.2.3 DAC

There are several different types of DACs, including parallel and serial types. Due to the timing constraints, as well as the availability of a data bus, the parallel type was chosen. This also simplifies the control logic.

The digital to analog converter chosen was an AD8582AR [56]. This DAC has the following features:

- Two channels
- 12-bit resolution using parallel interface
- 0-4.096 Voltage output range
- 2.5 Volt reference voltage
- 16 microsecond settling time

The DAC is interfaced to the peripheral bus of the controller. The DAC is a 5 Volt device. To make the peripheral bus compatible for devices that are not 5 Volt tolerant, a

LVCMOS buffer (Texas Instruments SN74LVC16244-ADGGR) was added between the DAC and the actual bus itself. All signals come from the FPGA.

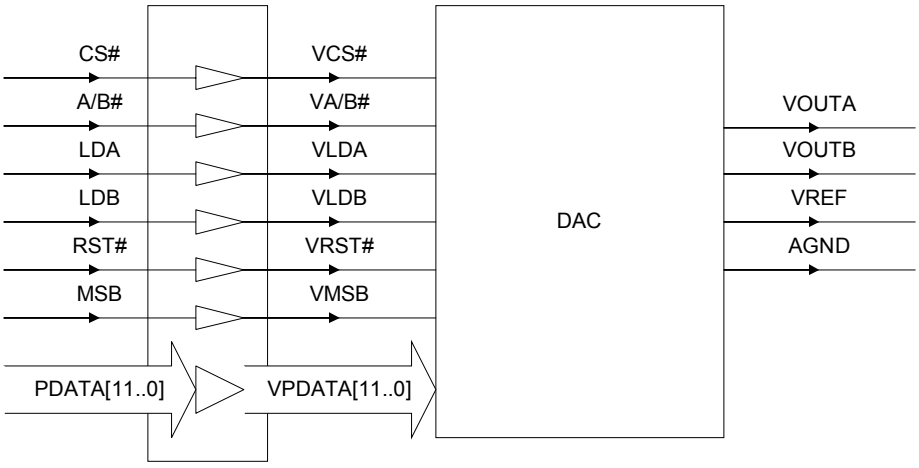


Figure 39: DAC Block Diagram

Since the DAC is purely an output device, and sends no control or status signals back to the FPGA, the output enable pins of the LVC16244 buffer could remain active at all times. The chip ignores the signals until the appropriate control signals are sent.

The control for the DAC consists of both types of data: that which is sent from the DSP, and lasted the duration of the transaction (referred to here as Transitory Data), and that data which is latched in the FPGA and remains after the transaction (referred to here as Persistent Data). The persistent data for the DAC is the RESET signal, RST#, and the MSB signal, which decides during a reset which value the DAC should reset to: 0x000 (0 VDC) or 0x800 (2.048 VDC). The control register for these two control signals resides in the control register block within the FPGA. The actual data that changes the value of the analog channels (transitory data) is routed using the DAC block.

The DAC block primarily functions as a timing controller for latching the value into the DAC. The setup and hold times must be valid for the DAC to recognize the new value. The DAC block uses a state machine that allows it to ensure that the data is working correctly.

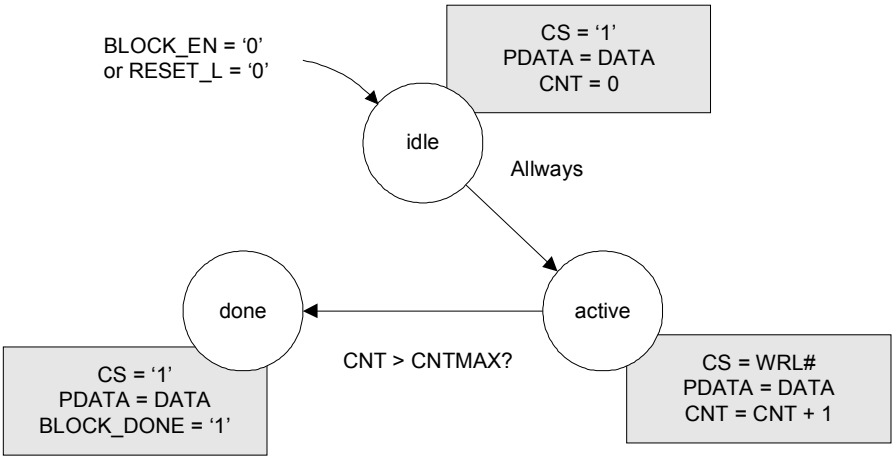
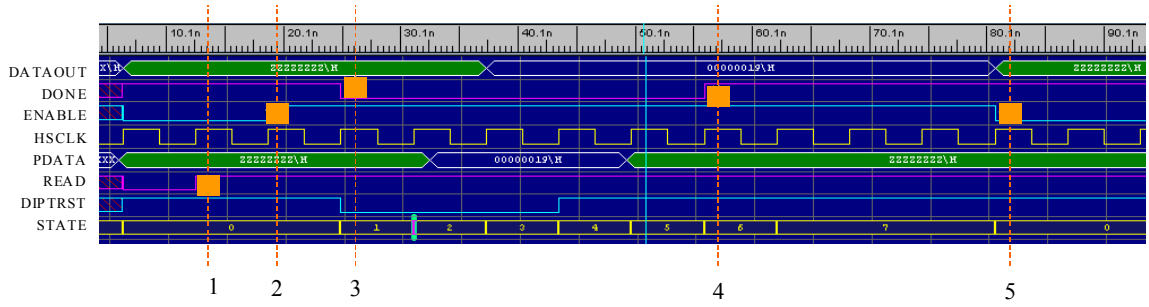


Figure 40: DAC State Machine



TIME	EVENT
1	ADDRESS FROM DSP DECODED AND BLOCK INPUTS SET UP FOR DIP SWITCH
2	SELECTOR ENABLES DIPSWITCH BLOCK
3	DONE GOES LOW, INDICATING BLOCK IS WORKING. DURING THIS TIME, THE BLOCK ENABLES DIP SWITCH OE AND UPDATES BLOCK_DATA_OUT PORT IN FPGA
4	DONE GOES HIGH, INDICATING BLOCK IS FINSIHED, AND DATA IS AVAILABLE OR HAS BEEN PROCESSED
5	DSP DISABLES BLOCK, AND BLOCK TRISTATES

The AD8582AR is packaged in a 28 pin SOIC package, with 50 mil pitch.

3.2.2.4 HEX Display

The hex display is used for debugging and visual indication of status. There are two digits displayed. During debugging, these digits can be used to represent a system variable, converter operating state, or set-point.

The TIL-311 [57] was chosen because it automatically decodes the four bit data into the correct pattern to represent the corresponding data on the display. No decoding is necessary to implement in VHDL, allowing the data to be directly passed from the DATA bus to the PDATA bus.

The hex display is interfaced to the controller via the peripheral data bus. It is a five volt device, and therefore, it exists on the VPDATA bus, which is the five volt unidirectional extension of the PDATA bus. There are two control lines that are used. The first one is the blank input. When this is high, the hex display does not show any digit, and appears blank. The second control line is the latch input. When this is low, data passes from the VPDATA bus into the hex display. Typically, the data is set up, and then this line is pulsed low to allow the data to propagate into the internal latches in the hex display. When it is high, the hex display ignores the values on the VPDATA bus.

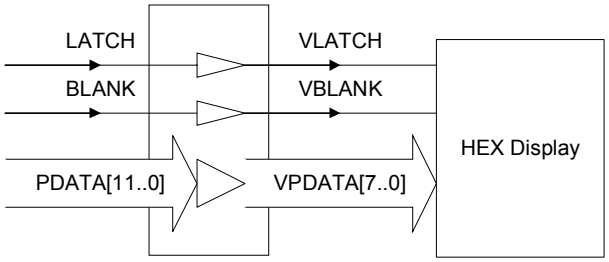


Figure 41: HEX Dispaly Block Diagram

The hex display is a memory mapped peripheral controlled from the FPGA. When the FPGA is addressed, and the selector determines that the address is the address of the hex display, it will set BLOCK_EN to high, and it will then wait for BLOCK_DONE to go low and then high. The following state machine shown in Figure 42 represents the control for the hex display. Data is sent on to the PDATA bus, and it becomes stable while the device is in idle. As soon as the device becomes selected, the state machine moves into the active state. The LATCH signal is pulsed low (inactive), allowing the data on the PDATA bus to propagate into the hex display. The hex display will remain in this state until the counter expires, indicating that the setup and hold time prescribed in the datasheet has expired. After this, the state machine moves into the done state, setting

BLOCK_DONE to high, and setting LATCH back to high, freezing the data. The device returns to the idle state when the selector deselects the hex display control block.

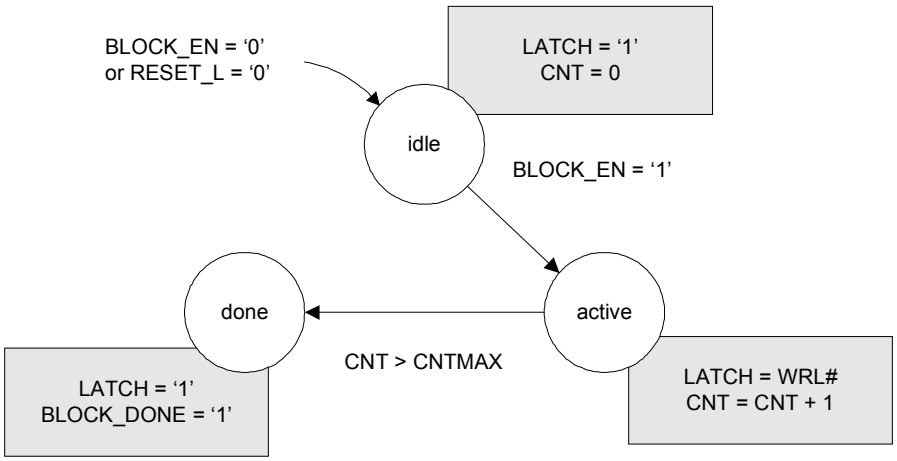


Figure 42: HEX Display State Machine

The HEX display is in a DIP-14 package. In order to make it surface mounting, a SMT DIP socket was used.

3.2.2.5 DIP Switches

DIP switches are useful for setting parameters that can be used as input to control code in the DSP. The controller has eight user DIP switches that are general purpose. The DIP switches were interfaced to the peripheral bus due to a shortage of pins. In order to interface them to the bus, a tristate buffer was used. When the PDATA bus is tristated from the FPGA side, the buffer can be enabled to allow the DIP switch data to propagate to the bus.

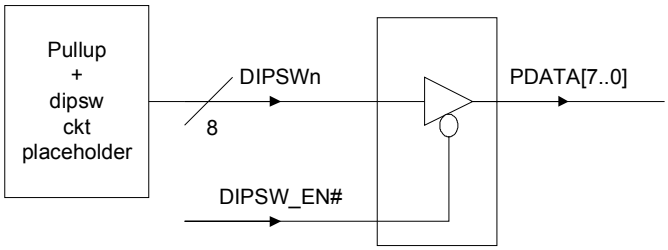


Figure 43: DIP Switch State Machine

The DIP switch state machine works similar to the hex display, except that it must latch the data once it is available on the hex display. The state machine will not enable the tristate buffers if the DSP does not request a read.

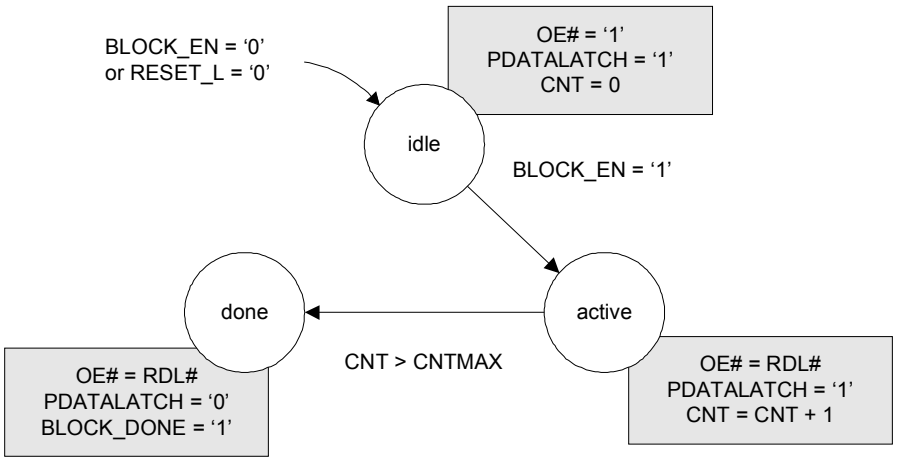


Figure 44: DIP Switch State Machine

3.2.2.6 PCI Mezzanine Interface

PMC, or PCI Mezzanine Card, is a standardized interface that consists primarily of the signals available on a PCI card. This type of card enables the controller to be physically embedded into another system. The choice to use the PMC form came from the use of this type in hosts other than PCs, such as VME and CompactPCI.

The PCI Mezzanine Card interface is described in IEEE p1386.1 [58], and is a daughter specification to the CMC (Common Mezzanine Card) specification, IEEE p1386 [58]. The CMC specification defines the size of a double-wide mezzanine card to be 149mm x 149mm. The Universal Controller was chosen to be a double-wide mezzanine card because it was impossible to fit all the contents of the controller into a single-wide one.

The PMC plugs into a host carrier card that interfaces it to the host that the PMC plugging into. The host carrier card is specific to the host type that the PMC is in. For

example, in a PC, there are host carrier cards to plug a PMC into the PCI bus. In a VME system, there is a carrier card chipset that can take a PMC, and interface a VME system via a bridge chip. There also exist similar ones for compact PCI and other architectures. This decision was made so that there was no restriction on the host system type.

The PCI Mezzanine Card Interface can use up to four connectors per slot, referred to as JP1, JP2, JP3, and JP4. JP3 and JP4 are used as I/O connectors. In the Universal Controller, these two are unused. The controller instead uses JP1 and JP2, which carry the PCI signals to the host carrier card. Power for the controller is also taken from these two connectors when the card is in a host. Jumpers J7 and J8 are both set to PMC to enable the power to come directly from the PMC connector instead of from the power connector (J6).

The chip chosen to implement the PMC logic (mostly same as PCI) was the Cypress CY7C09449PV-AC, also referred to as the PCI-DP chip. The function of the chip is to implement a dualport interface between the PCI bus and a localbus. In the case of the controller, the localbus interface is the PDATA and PADDR busses connecting to the FPGA.

The CY7C09449PV-AC uses a dual port RAM interface to the PMC bus. Dual port memory is memory that can be simultaneously accessed from two different sets of address and data lines (referred to as ports). This memory type is typically used at bus interfaces or at interfaces between processors to exchange information needed by both systems. A dualport ram configuration is shown in Figure 45. The busses may not even use the same protocol, such as in the case that the PCI-DP is. Dualport interfaces are also used in industrial fieldbus configurations for sharing information between independent systems for reasons such as compatibility and fault tolerance [59].

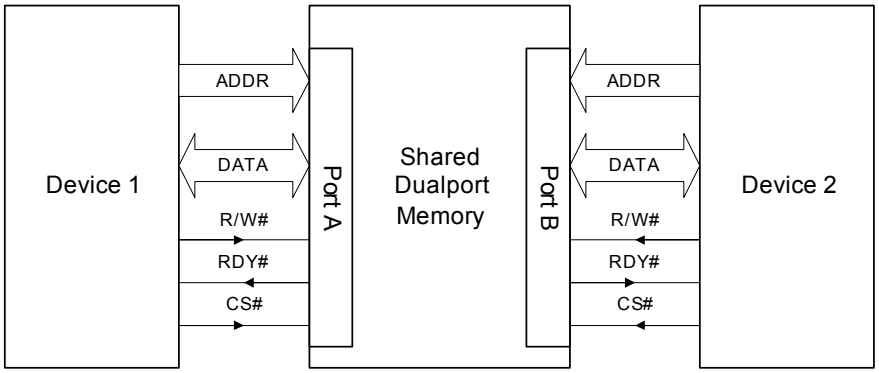


Figure 45: Dual Port RAM Example

The chip has a localbus configuration register to modify the functionality of the localbus interface. The VHDL code was written to use the default configuration. If any initial parameters need to be modified, an I^2C interface is available. On power-up, the chip will try to load the PCI configuration register values and the localbus configuration register values from an I^2C compatible EEPROM. These pins are made available to the FPGA, but for the tests done, these pins were not used.

The chip has Address lines 14 down to 2, and then has four byte enables. The lines are connected to the FPGA starting with the four byte enables at address lines 0 through 3, and then address line 4 is byte enable 2. Therefore, to write to a memory location in the PCI-DP chip, the address has to be shifted to the left by 2 first. Writes to this chip have an address phase and a data phase. When the chip is accessed, the address is first placed on the bus, SELECT# is set active, and then STROBE# is set active. To exit the address phase, STROBE# is deasserted, while SELECT is left asserted. Upon the completion of the operation, the PCI-DP will assert RDY_OUT#. At this point, the data can be captured if it is a read, and the transaction can end with SELECT# deasserting.

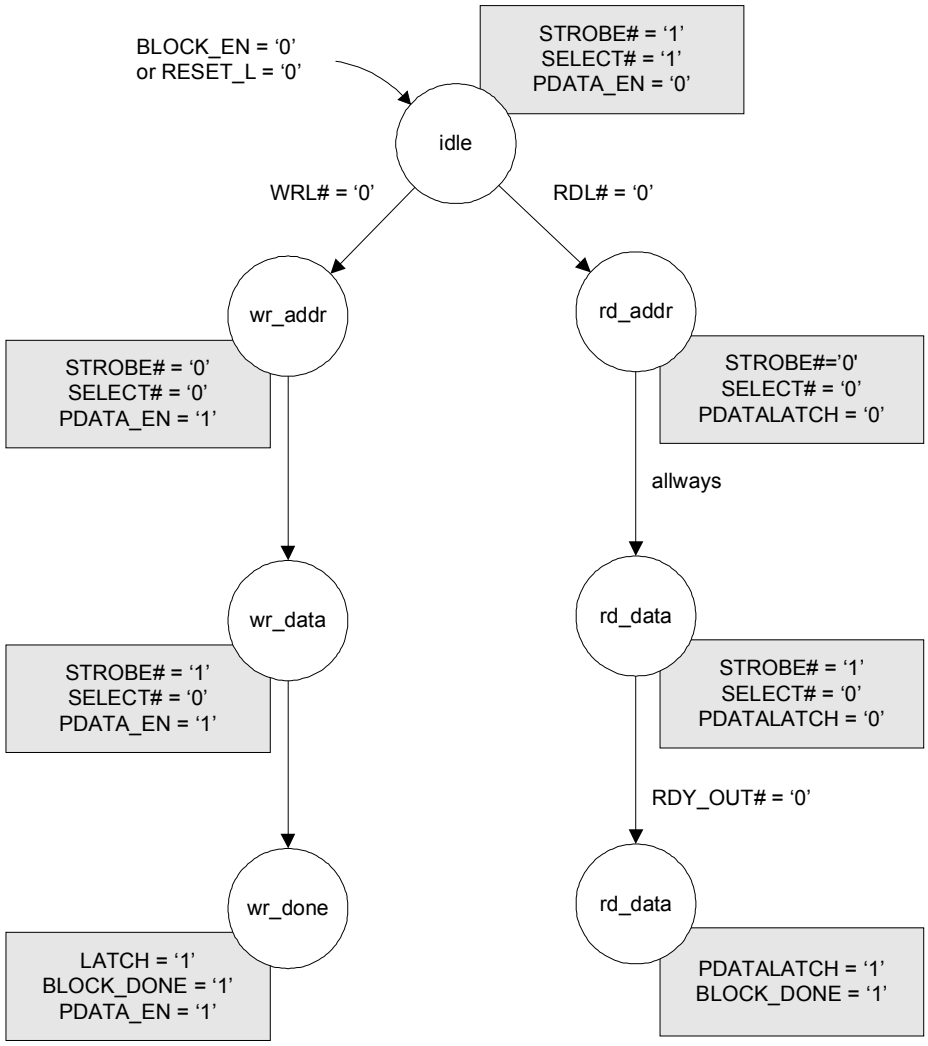


Figure 46: PMC State Machine

Initially, there was no response from the Cypress chip when a command was given. Originally, the FPGA provided the clock to the Cypress chip. The chip needs a clock on the localbus as soon as it is powered on, however. To fix this, a buffered copy of the PCI clock was fed back into the chip localbus clock, and a copy was sent to the controller. The controller synchronized its bus operations with this new clock. Since the new clock (PCI 33 MHz) was much slower than the FPGA clock (80 MHz), it was possible to "poll" the cypress chip clock to detect a rising edge. When the clock is low, a lock is set to 0. When the clock is sampled at '1', the state machine transitions, and sets the lock to 1.

For every following sample, there will be no transition, as the lock is set. The only time that the state machine transitions is when the clock is 1 and the lock is not set.

To verify the PMC was operating correctly, and to facilitate easier debugging, the controller was placed in the PC. Since the fiber optics were working, it was easy to use the fiber in this debugging process. The process used two controllers. The first controller (the one under test) was placed in a host PC running Windows CE (referred to as a CEPC [60]). The second controller was connected to a development PC, and was attached via the JTAG emulator. Using the development PC, it was possible to send commands to the controller under test in the CEPC using the fiber optic interface with PESNet 1.2.

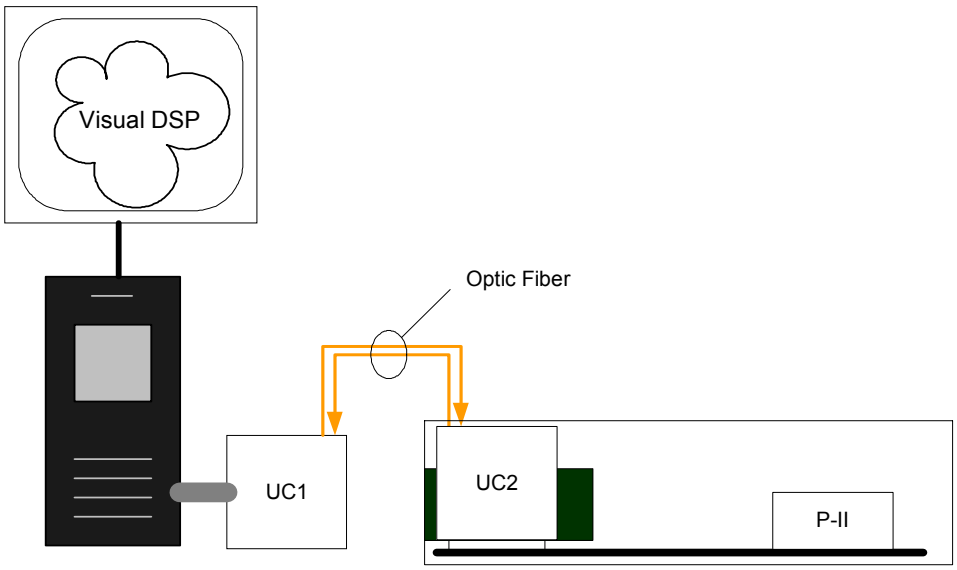


Figure 47: PMC Test Setup

The PESNet registers consisted of three 32-bit words in and three 32-bit words out. The first register contained a key and an operation. In order to control when operations occurred, a key was used as part of the message sent to the card under test. The second program would not execute anything until the key changed. Two operations were possible (a read from memory and a write to memory). The first field contained the key

and the operation as 16-bit fields. The second register contained a pointer to the memory address to modify. The third address contained data to write in the case of a write operation. This procedure is summarized in Figure 48. The process in interrupt driven, and occurs every 20uS.

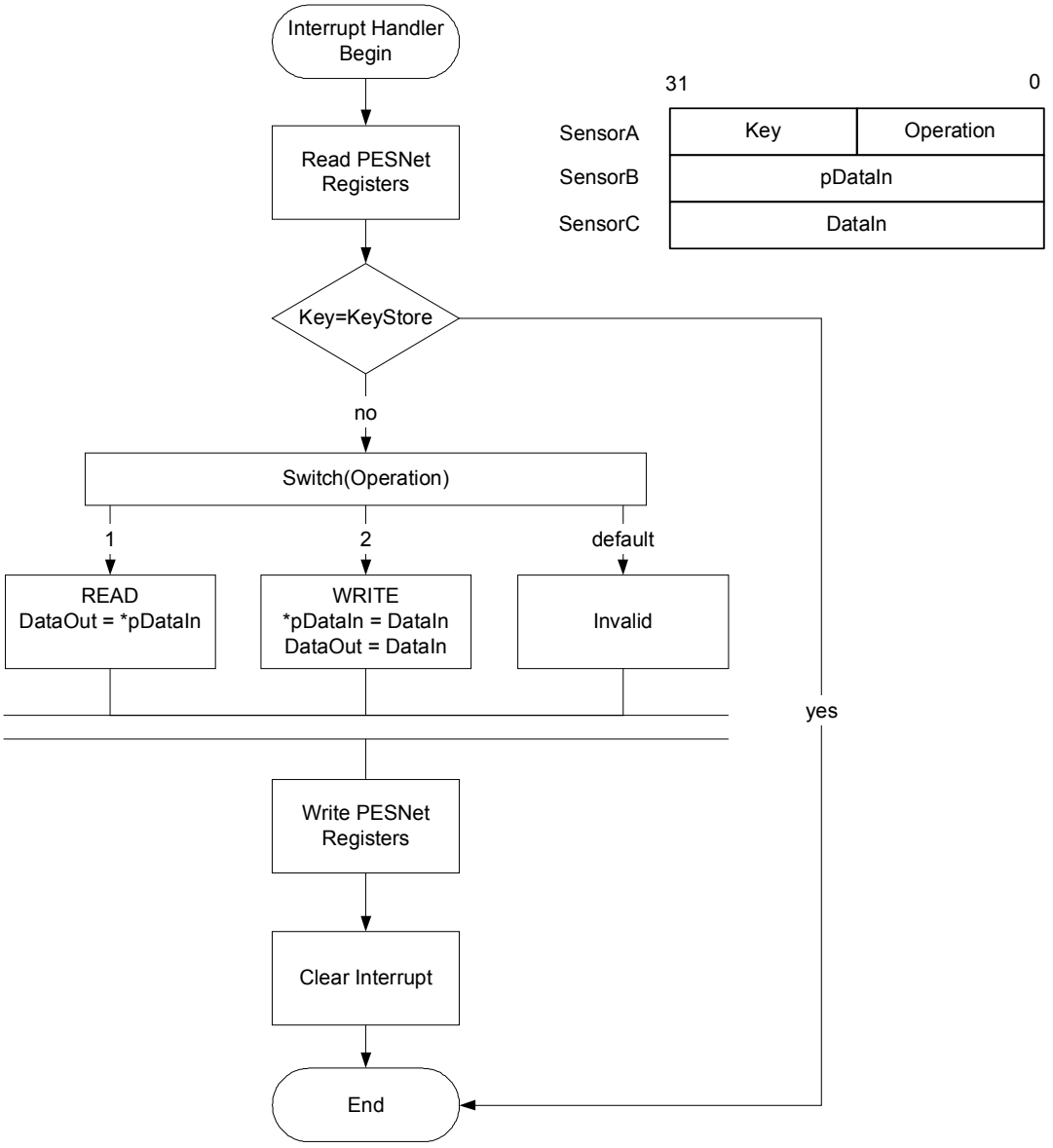


Figure 48: PMC Test Remote Code Flow Chart

The program was stored in the DSP boot flash, and was loaded when the DSP came out of reset from power-on.

3.2.2.7 DSP Boot Flash

When the controller is powered on or reset, the DSP will use this flash to load instructions and execute control code. While selecting flashes, it is desirable to have one with the following properties:

- Boot Sector
- Sector erasable
- Standardized flash interface

The flash chosen was the AMD Am29LV040 [61]. This flash resides on the DATA bus lines [38..31] as specified by Analog Devices [62]. There are three control lines going to this device that are the equivalent of a read enable, a write enable, and a chip select. The block diagram of the DFLASH – DSP interface is shown in Figure 49.

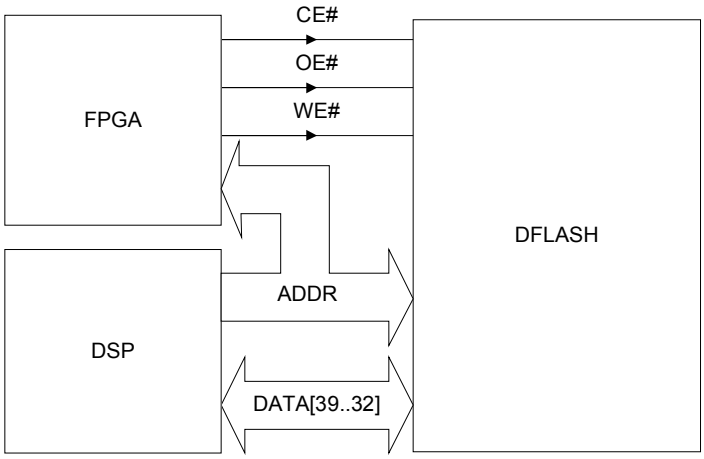


Figure 49: DFLASH Interconnection Diagram

According to the DSP hardware reference [63], the DSP addresses the flash via the use of the BMS# pin. This pin is also used to program the flash. However, it is complicated to use this pin, and even the code shipped with the demo board for the ADSP-21160 uses a different method. To take advantage of this already-available code, while still allowing the DSP to boot via BMS#, two methods of activating this flash are implemented. The flash can be activated by addressing a memory location in the range assigned to the flash. This is used to erase and program the flash, as well as to read data from it later in the

program. The BMS# pin is asserted by the DSP on startup, and this method is equivalent to selecting the upper address bits to match the range assigned in the FPGA memory map. The demo code shipped with the ADSP-21160 flash, called EZ-KIT [64] was modified for the AMD chip. The code was originally written for a ST-Micro flash, which used slightly different commands for writing and erasing.

3.2.2.8 Peripheral Flash

It is desirable to store data in a flash for diagnosis purposes. After a fault, the flash would provide a time history of the states. Other configuration information could be stored here as well. This flash is similar to the boot flash with the exception that it has no boot sector.

3.2.2.9 Fiber Optic Interface

Each fiber optic transceiver is connected to the FPGA directly. Described here is the interaction between the cypress chip and the FPGA at the physical layer. A description of the protocol itself is discussed in Chapter 5.

The fiber optic transmitter chips have several connections to the FPGA. These connections are shown in Figure 50.

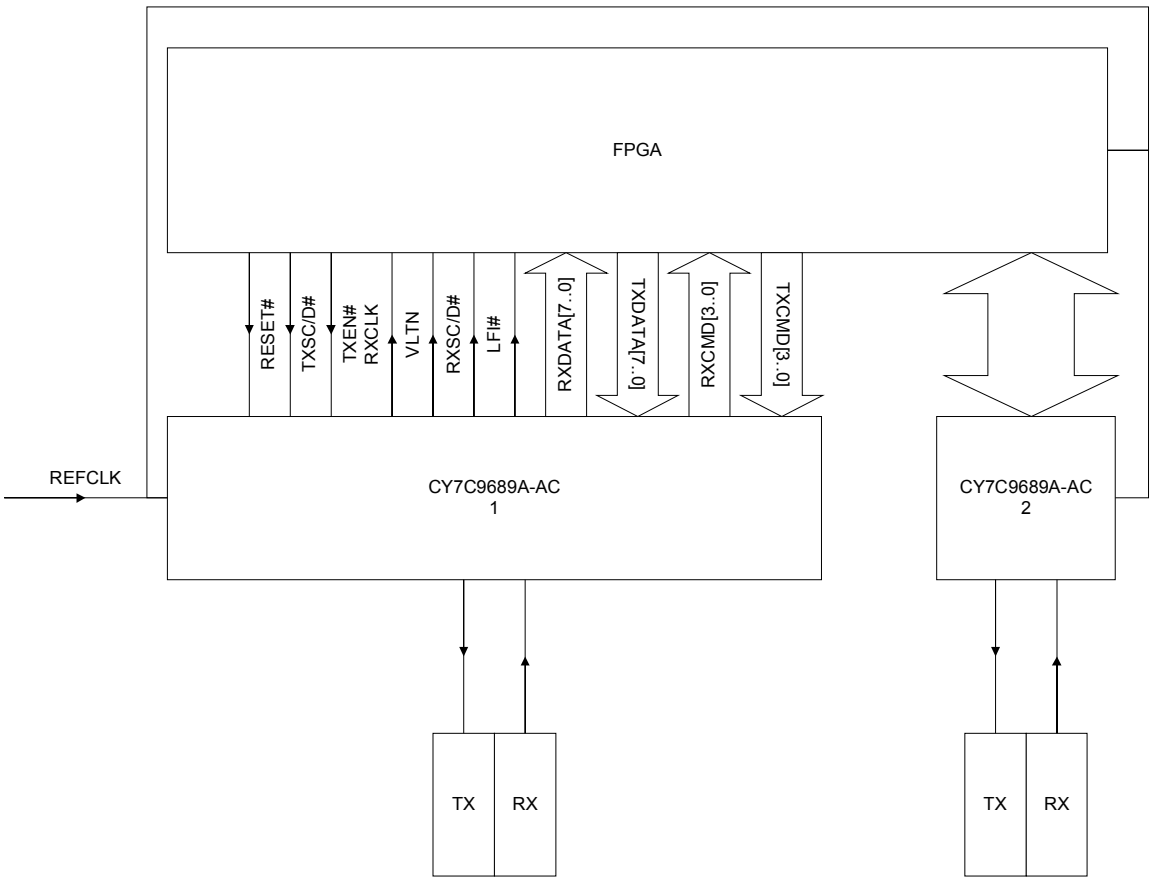


Figure 50: Transceiver Chip Interconnection Diagram

The cypress chip has many different possible modes of operation. It has built-in FIFO buffers for storing communication packets, and status flags indicating that these buffers are full, half-full and empty. However, in the mode that the controller operates in, these flags take on different meaning. The controller uses the cypress transceiver chips as synchronous elements, as the nature of the communication protocols supported are synchronous. These flags toggle at half the speed of refclk, which operates at twice the speed of the character clock. When the empty signals go low, it is possible to change the data on the chip interface, and when the empty signal goes high for the receiver (inactive state), valid data can be read from the receiver if the fault flags are inactive.

The chips can transmit either command characters or data characters. There are command inputs and outputs, as well as data inputs and outputs available to the FPGA from the chip. If the FPGA wishes to send a command, it will assert the TXSCD line,

indicating that the command bus contains valid data. Similarly, the transceiver chip will assert this line indicating that it has received a command character. If the character is a data character, the respective line will be deasserted.

There are violation and link fault indicators to indicate that the in-chip PLL cannot recover the clock from the serial stream presented to it from the optical to electrical receiver. Such a fault may indicate that either there is a fault on the PCB, the fiber is not plugged in, or the other node sending the data is not powered on or is misconfigured. If a bit error occurs, the violation line will rise, indicating that an invalid pattern was received on the fiber, and that the character is invalid.

There are four DIP switches used to configure the behavior of the communication chips. These four are located at the locations described in Table 2:

Table 2: Communication Chip Configuration Switches

Location	Name	Description	Normal Jumper Setting
J1-1	DLB	Diagnostic Loop Back	ON
J1-2	BISTEN	Built-in Self Test Enable	OFF
J1-3	RFEN	Reframe Enable	OFF
J1-6	ENC	Encoder Bypass	OFFF

The protocol supports a limited number of nodes, and is very similar to that described in [65] and in [66].

3.2.2.10 Peripheral Expansion and Debug Connectors

The peripheral expansion and debugging connectors serve the following functions:

- Debugging VHDL modules
- Interfacing boards that expand the functionality of the controller
- Providing an interface into the system for a logic analyzer
- Bus Monitor

Several examples of expansion would be fiber optic transmitters and receivers used to control peripherals local to the controller, such as crowbars or safety devices. It can also be used to generate PWM signals to control things such as analog and digital meters.

They can also be used to implement serial busses to support peripherals such as LCDs.

The peripheral expansion and debug connectors are pins directly connected to the FPGA. These pins can be programmed by writing custom VHDL modules that will control their behavior. These blocks interface to the control register or they receive commands from the data and address lines, enabled by the selector.

One example where this was used was in the verification of the analog to digital converter for the hardware manager. The code was developed and debugged in simulation before the board was ready. Instead of waiting for the real hardware manager to come back, the analog to digital converter control block (written in VHDL) was placed in the controller, and interfaced to the peripheral connectors. It was easy to create an in-house PCB on which the ADC could be placed. Here, the block was verified. When the hardware manager came in, the ADC operation was already guaranteed.

3.2.2.11 Global Control Connector

A connector was made available to interface a daughter card that supports a higher level communication protocol for several purposes:

- Supervisory control of the converter (commanded from a PLC)
- Monitoring of the converter
- Adding distributed periphery to the universal controller (controller is busmaster)

This global control connector is interfaced to the peripheral bus, and supports 3.3V and 5V devices. The 3.3V devices do not have to be 5V tolerant, as the 5V section of the bus is isolated using buffers. Several vendors make chips to support upper level communications. Some of these are listed below.

Table 3: Fieldus Implementation Chips

Protocol	Vendor	Chip	Description
Profibus-DP	Siemens	SPC-2	Supports FDL
Profibus-DP	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components
Profibus-FMS	Siemens	ASPC-2	DSP Interface ASIC
Profibus-DP	Profichip	VPC3+B	Slave ASIC
LonWorks	Cypress	CY7C53150	DSP Interface ASIC
DeviceNet	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components
ControlNet	Allen Bradley	CAN 10	DSP Interface ASIC
AS-i Bus	ZMD	A2SI-ST	AS-i master
Industrial Ethernet	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components

The Global Control Connector has all of the peripheral address and data lines available, as well as the bus control lines. In addition, there are ten lines private lines that are customizable depending on the ASIC that is used on the daughter card. Some of these chips use serial interfaces, in which case two or three of these lines will be used for the interface, and the bus interface is not used at all. The interface code is written in a VHDL module, and placed into the FPGA modular architecture, removing the placeholder for the global control interface.

3.3 Methodology

The following section describes the procedures used to implement the Universal Controller.

3.3.1 Universal Controller PCB Design

There are many aspects of implementation that are not considered when designing the logical interconnection of components as in the previous steps. With so many components sharing the same signals, the layout of the controller needs to be considered carefully. The FPGA has 560 connections to the PCB itself within a 3x3cm area. The

DSP has 400 within a 2.7cm x 2.7cm area. Many of these signals are switching at 40 or 80 MHz. It is important to consider the distance that these signals have to travel.

Another consideration is EMI shielding. Several components are noisy, and should be shielded so that the noise does not affect other signals on the PCB. With so many components switching at different frequencies, bypassing and power planes become important to consider. Having planes, in turn leads to issues such as copper balance vertically throughout the PCB.

Another constraint to the PCB design is the mechanical layout specified by the CMC standard (PMC parent specification IEEE 1386 [58]). This specification requires the PCB to be 149mm x 149mm with a thickness of 62 mils. There were both good and bad effects of this specification. The good effect is that the signal lengths were reduced, eliminating problems introduced due to long traces. The bad effect is the increased density of signals. The higher density of signals led to blind vias between three layers: Top and Midlayer 1, Top and Midlayer3, and Bottom to Midlayer 6, as shown in Figure 51. This added to the warpage of the PCB, and dramatically increased the cost.

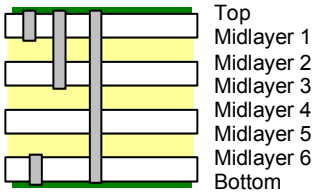


Figure 51: Layer Stack

Another important area of the PCB was the optical transmitter. This area had very sensitive signals switching at 125 Mbps. This area is sensitive to noise, and so several precautions were taken to ensure its successful operation. First, an exact layout of the transmitter was given by Agilent Technologies, the manufacturer of the optical transmitter and receivers in application note 1066 [67]. The GERBER files were obtainable for this circuit. Although it was not possible to directly apply the GERBER files to the circuit, the layout could be copied point by point for every trace to exactly

replicate the layout twice in the controller design. Since the design of this transmitter/receiver circuit used 4 layers, and the controller used 8 layers, the additional layers were replicated as power planes to increase the noise immunity and add capacitance between the isolated power plane pair.

The final version of the controller had a total of 709 nets, which were connected to a total of 3280 pads. Additional characteristics of the PCB are listed in Table 4.

Table 4: PCB Attributes

Attribute	Value
Number of Layers	8
Number of Holes	1978
Number of Vias	1912
Number of Components	396
Smallest space between different nets	5 mil
Smallest trace width	5 mil
Mask Type	HASL
Number of Nets	709
Number of pads on top	2049
Number of pads on bottom	1143

While the PCB software, Protel 99SE, had autorouting capability, it was insufficient for such a dense 8-layer PCB. With the help of an external contractor, Gasha Gataric, the first version of the controller was manually routed. The final PCB is shown in Figure 52 and Figure 53.

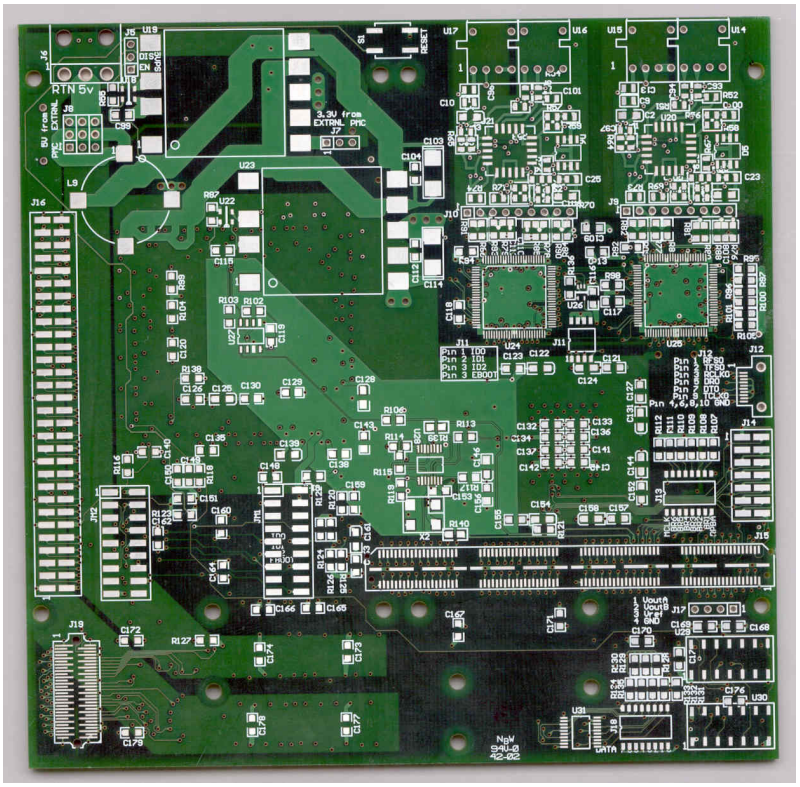


Figure 52: Universal Controller PCB Top

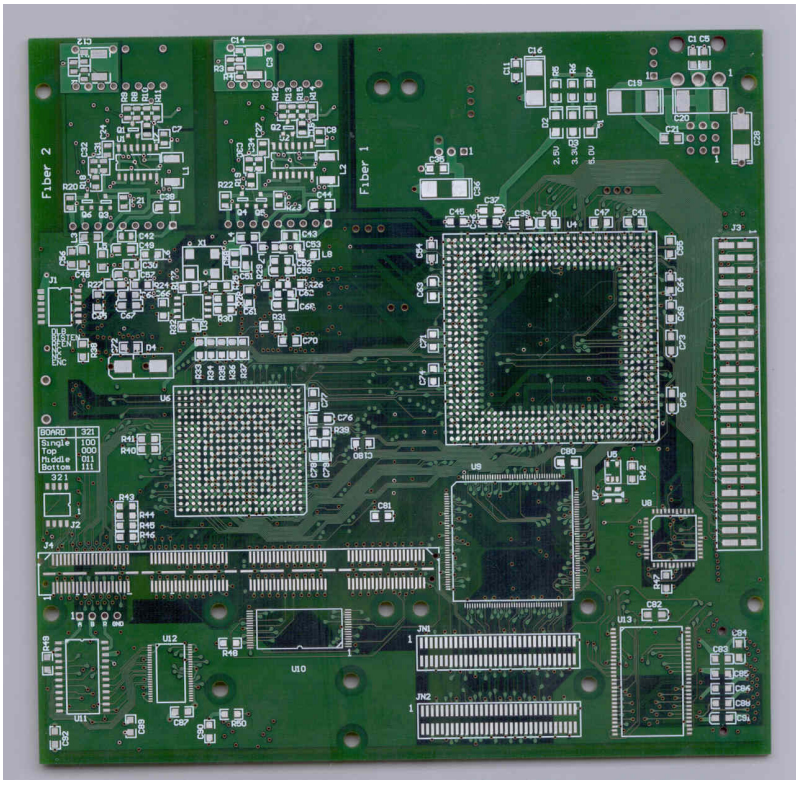


Figure 53: Universal Controller PCB Bottom

3.3.1.1 Quality Assurance

During the development cycle, there were several points at which the controller was examined by people other than the developer. This is important, as there are implicit assumptions made by the designer that are impossible to recognize by someone who is looking at their own work. To address this, quality teams were assembled to examine the controller for defects in design at several stages. The teams consisted of electrical engineering, computer engineering, and computer science B.S., M.S., and Ph.D. candidates. Methods used for quality assurance came from previous work experience in a software development company, as well as from [68], [69], [70], and [71].

QA sessions were held after the completion of the schematics, but prior to the PCB design or modification. Two major sessions were held. The first one was during the design of the first board, and the second one was prior to submitting the modifications for the second board to the PCB manufacturer. Checklists governed the objectives of each QA session. Two teams of two people were assigned to each area of the reviews to avoid group think behavior probable with four people on a team while at the same time double-checking everything. There were several areas to review during each session.

First, the schematic symbols had to be checked. Each schematic symbol was created from the component datasheet. QA teams checked that each pin of each component matched each name on the schematic symbol. A discrepancy at this point would propagate through the rest of the design, and would be very hard to fix in the future.

The second type of check was to make sure that the pad size and location of each PCB footprint matched the specifications on the datasheets. If the footprint was wrong, then the component would not fit after manufacturing. It is also important to check that Pin 1 was in the same location, as some packages, especially TQFP and PLCC types vary the location of pin 1 from component to component.

The third type of check was a netlist check. It is easy to misspell a net name, or to use different notation in different locations when they should be the same. Examples of this are: +3.3V vs. 3.3V vs 3V3 vs VCC. Similarly, active-low signals have the same problem: BMS, \BMS, nBMS, BMSn, BMS_L. Other issues that can occur in this area are nets that are the same, but they are still separated due to some aspect of the PCB software, Protel 99SE.

For the PCB check, several issues were examined:

First, the mechanical dimensions were checked. The CMC specification has several holes for the double-CMC form factor that specify the x-y location as well as the diameter and clearance for each hole.

Secondly, aspects of the routing were checked for undesirable features, such as multiple vias in a single trace, as well as traces going past noisy areas of the PCB, such as oscillators and power supplies. Noisy traces such as clocks had guard traces or planes to shield these components. Other sensitive areas were the communication transceivers, which transmitted at 125 Mbps, as well as the transceivers and FPGA.

Thirdly, the power planes were checked to ensure that they were connected correctly. In the second PCB check, the copper balance was also checked. A significant defect in the first design was the copper imbalance from top to bottom of the PCB. If the copper is not balanced, the PCB will tend to warp and twist, which in turn makes large surface mount components difficult to solder, especially BGA (ball grid array) devices such as the DSP and FPGA. One PCB failed assembly due to this issue, as the X-RAY revealed bridging between two connections on the corners of the BGAs. This is discussed more in the issues area.

Many of these QA issues resulted from the initial inspection of the PCB and the resulting tests. They were added into the second revision, as the issues were recorded in a database, which in turn created a checklist of issues to explicitly verify in the new design.

As issues were found, they were added to a quality database to make a complete history of the issues found in the PCB, who found them, when, what category they were related to, and a history of comments associated with that issue describing what actions were taken to correct it. This database in turn automatically produced a release checklist that was reviewed to ensure that all issues were accounted for before manufacturing the second design. The use of a release checklist can be found in [68].

During several points in the design and development cycle, the design was presented to project stakeholders for review and comments. Using their feedback, the design was modified to include their input. Outcomes such as the use of PMC resulted from these discussions.

General Dynamics Advanced Information Systems conducted another PCB design review, and feedback to each change was discussed during weekly teleconferences that lasted from 30 minutes to 90 minutes.

The first revision of the PCB turned out very successful. There were some issues with equipment and software, which took time to fix, but once these issues were solved, there were not a lot of problems left, as the QA sessions identified many problems that would have rendered the PCB unusable if they were not identified.

3.3.1.2 Issue Tracking Database

It is important in complex designs to keep track of problems so that they may be fixed before the next release. These problems can then be used as inputs to future designs so that those designs avoid the same mistakes. A database was designed that would maintain a list of issues, as well the history related to that issue. This database facilitated the QA process.

The following fields were in the main table:

Table 5: Quality Database Fields

Field Name	Description
IssueNumber	A unique id that describes the issue
Issue Title	A brief description of the issue
Date Opened	The date that the issue was found
Date Closed	The data that the issue was closed
Assigned To	The person who is responsible for resolving the issue
Assigned By	The person who submitted the issue
Issue Type	**Issue:** A problem to be fixed **Assignment:** Something new that needs to be done
Issue Status	**Open:** Issue is submitted. **InProgress:** Issue is being worked on **Resolved:** Issue has been fixed **Closed:** Issue has been verified and closed. **Reopened:** Issue has been fixed, and verified, but now it is again a problem, or a canceled issue has become of interest again **Canceled:** Issue will be ignored
Issue Severity	**Critical:** Board cannot be manufactured with this defect, or it will be unusable **High:** Board can be manufactured with this, but the functionality will be severely impaired **Medium:** Board can be manufactured with this, but some features will be disabled **Low:** Board can be manufactured, but some features will not work exactly as designed **Cosmetic:** Board functionality will not be affected
Issue Priority	**High:** Issue must be resolved ASAP, because it will impede future work on the controller until it is addressed. **Medium:** Issue may have a complicated fix that should imply that it should be fixed before it becomes harder to fix as more work is done. **Low:** Issue is not of immediate concern.
Project	A description of which project this issue belongs to
Subproject	A description of which part of the project this issue belongs to. Ex: Hardware Manager, Universal Controller, PESNet, System, Cabinet, Logistics
Description	A more detailed description of the problem or how to reproduce it
Comments	As work is done, this field is updated to keep track of changes, status, and concerns associated with this issue. Each entry automatically places the date and submitter of the comment in the field.

3.3.1.3 Revisions and Final Design

After receiving the first design, feedback from the assembly house on the fabrication house, feedback from General Dynamics, and in-house testing, several modifications were planned for the second version of the PCB.

One of the significant improvements was the power plane bypassing capacitors. Originally, the capacitors had traces that made a circular path from the capacitor to the via. In the second design, the vias were moved much closer to the bypassing capacitor pads, and larger traces were used to connect to the vias. General Dynamics also suggested that the bypass capacitors should not be connected to the pins of the device. Instead, the purpose of the capacitors should be to bypass the plane beneath the device, and other vias should bring that power up to the component as close as possible. Several changes were made, especially next to the Cypress transceivers to accommodate these suggestions.

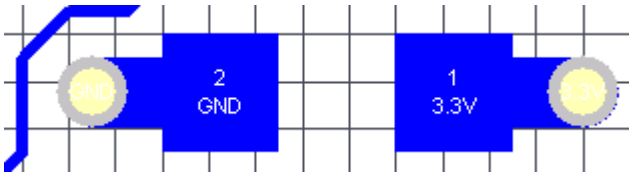


Figure 54: Improved Bypassing

The issue of copper balance significantly affected the PCB. To address this, additional copper, connected to the ground planes, was added to the perimeter of the PCB in the unused areas of the midlayers. Additional, larger planes were added to the top and bottom of the PCB, and were able to add shielding to clock buffers, oscillators, and the communication chips, as shown in Figure 55.

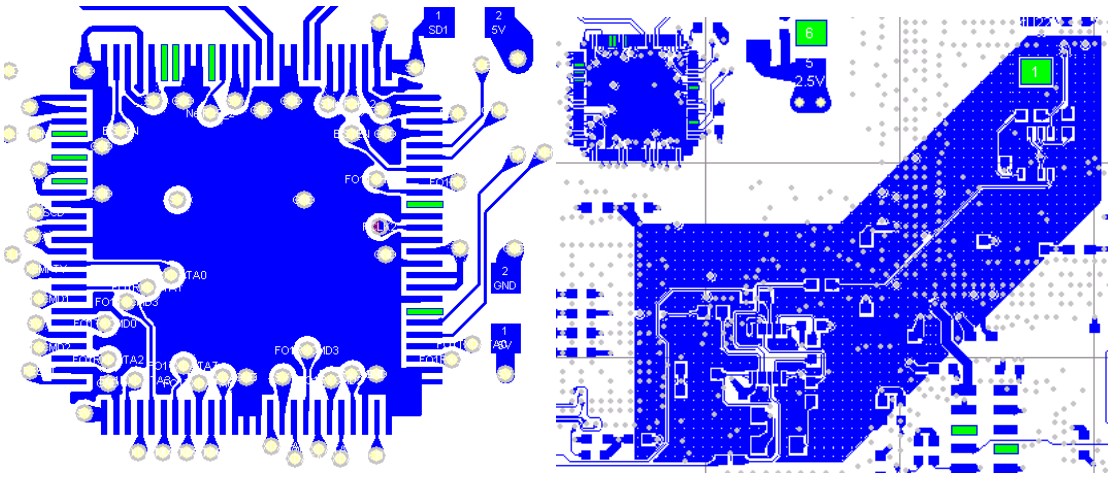


Figure 55: Shielding Planes

Based on feedback from the assembly house, the finish was changed from LPI to HASL, as it was easier to mount the BGA components. Other issues identified by the manufacturer include a lack of solder mask on some the top of some vias. This becomes a problem with BGA components, as it allows the solder ball on the BGA to migrate into the via hole, creating an open circuit between the device and the pad.

3.3.1.4 Final Design Modifications

After receiving the controller back, most functionality was verified. The fiber optic components required additional modification in order to ensure that they were functional. Two cuts are required on U24, and two jumpers need to be created, as illustrated in Figure 56. The two black lines indicate the jumpers, and the two red lines indicate the cuts. Note that this image is mirrored when looking at this side of the board. That is, U24 is to the left of U25 when the communication chips are facing up.

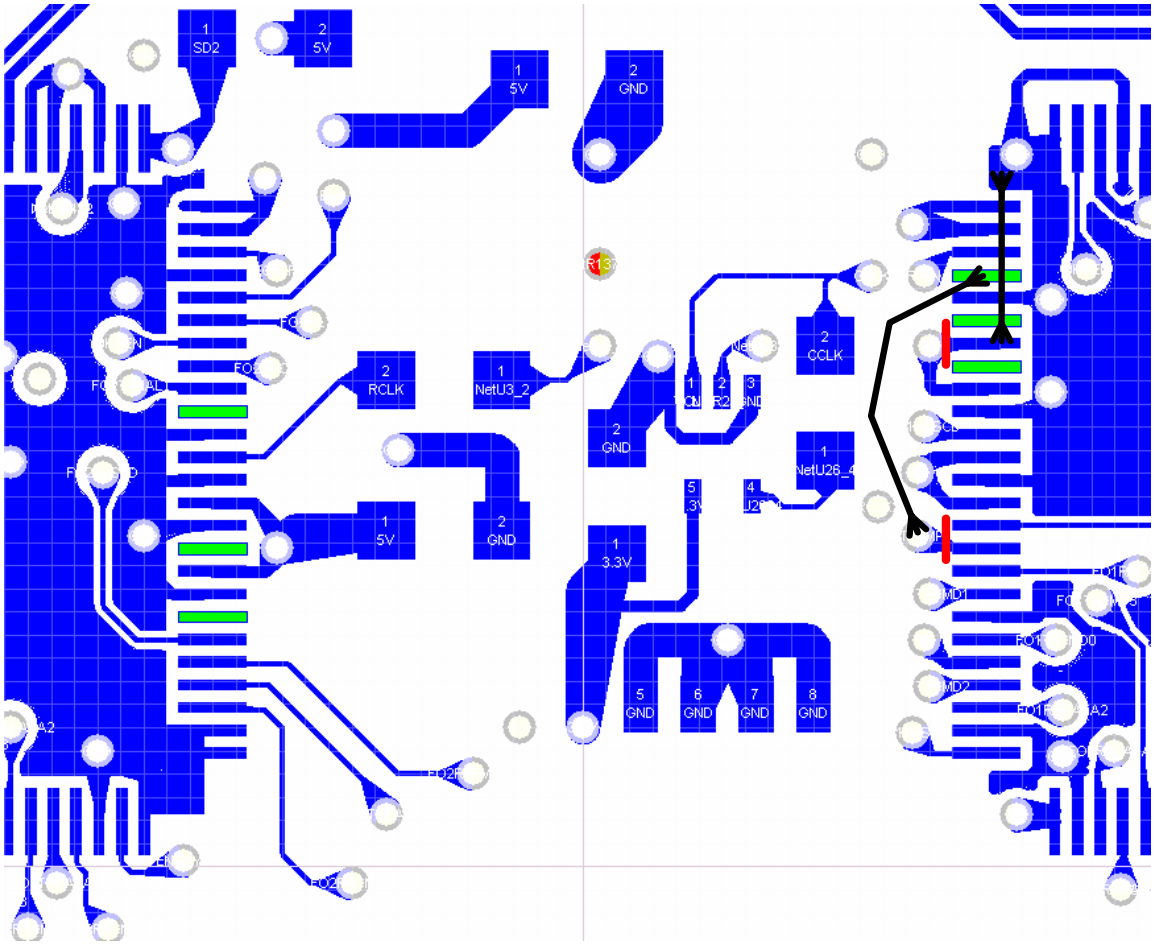


Figure 56: Modifications to U24

Another modification involves the CY7C9689PV-AC PCI-DP chip. The clock source for the localbus was changed to use the PCIPCLKOUT pin fed back into CLKIN. There will need to be a corresponding change in the artwork for the PCB for future designs.

4 Hardware Manager

The hardware manager is a distributed node that supports power level specific hardware. A previous version of a hardware manager was built and tested in [66]. The new hardware manager improves on this concept, but lacks the soft-switching capability of the previous hardware manager.

The structure of the power electronics devices that the hardware manager is supposed to control is shown in Figure 57.

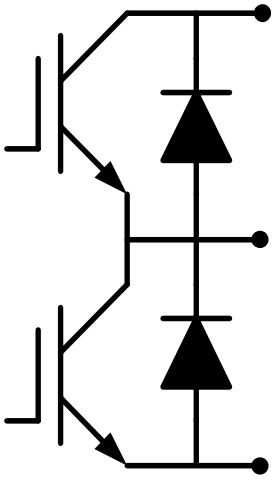


Figure 57: Half-Bridge Configuration

4.1 Design Requirements

The hardware manager (HM) should support the following features:

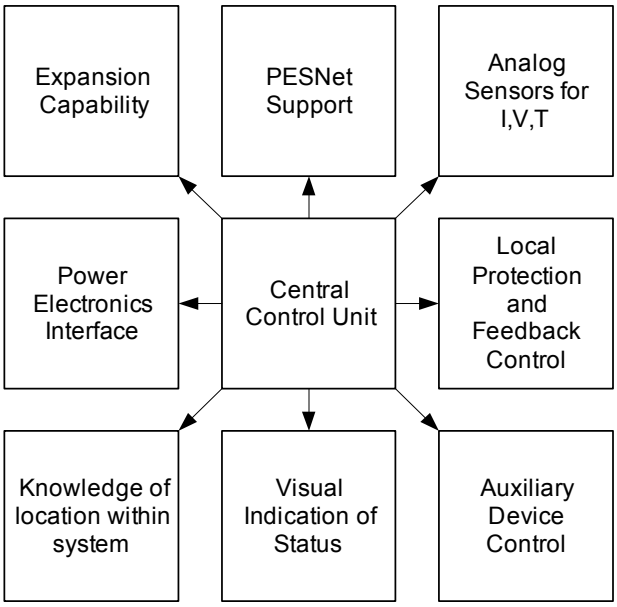


Figure 58: Hardware Manager Requirements

PESNet Communication Network Support. Being part of a larger, integrated system, the hardware manager should be able to send and receive commands from other nodes. The integrated communication network should be able to control the hardware manager timing, allowing all hardware managers to act synchronously.

Visual Indication of Status. The hardware manager should be able to display its status to people at a glance. A failed hardware manager should appear different than an online one. The user should be able to determine the type of failure from the visual information provided.

Measurement of Analog Variables. The hardware manager should have access to the state variables of the environment in which it is operating. These variables are used in protection and control. These are either acquired via PESNet or directly measured via sensors onboard the hardware manager itself. Variables of interest are the midpoint current, the DC link voltage, and the device temperature. Other temperatures on the

hardware manager are also important for safety and protection purposes. There is a resistor necessary for the voltage sensor that gives off a lot of heat. If this resistor gets too hot, then it may affect the operation of the rest of the PCB. A sensor near this area will measure the PCB temperature, and determine if it is too hot, in which case the FPGA should place the hardware manager in a fault state.

Power Devices Interface. One of the major goals of the hardware manager is to control the switching device that it is attached to it. The hardware manager is responsible for using the information given to it over PESNet, and possibly information from the sensors to produce control signals for the top and bottom switches of the connected IPM. The IPM is a packaged power electronics module. In this case it is in the form of a half-bridge. The IPM block diagram is shown in Figure 59. There are four digital terminals and three power terminals.

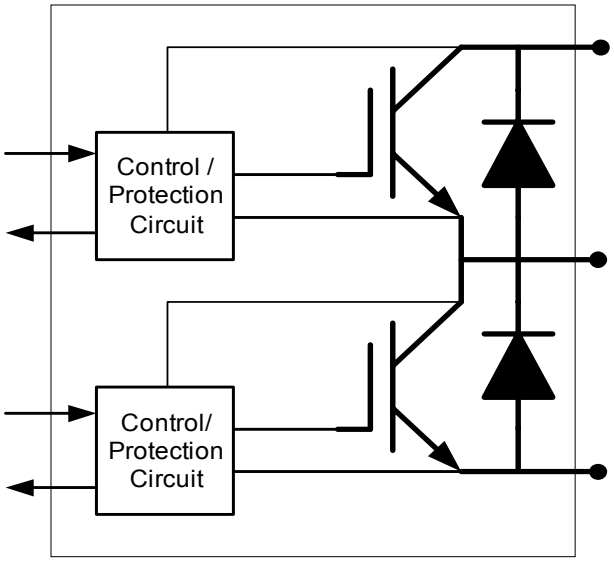


Figure 59: IPM Block Diagram

Although the hardware manager was designed for a specific power module, it should be able to be easily be adapted for use in another system, such as an ETO [72] (emitter-turnoff thyristor) based STATCOM [73] for utility applications, where each hardware manager would control five high power switches, contactors, and multiple sensors. The block diagram of an ETO is shown in Figure 60.

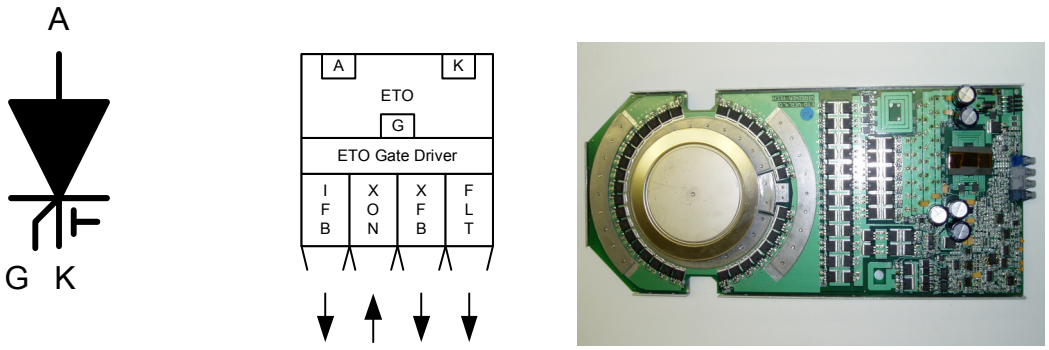


Figure 60: ETO: Schematic (left), Block Diagram (middle), Physical (right)

An input to the specification of the hardware manager should be an I/O analysis that determines the number and type of variables required to support an application. The digital requirements are listed in Table 6.

Another building block is an H-Bridge building block (referred to as a HBBB). The schematic of an HBBB (snubberless) is shown in Figure 61.

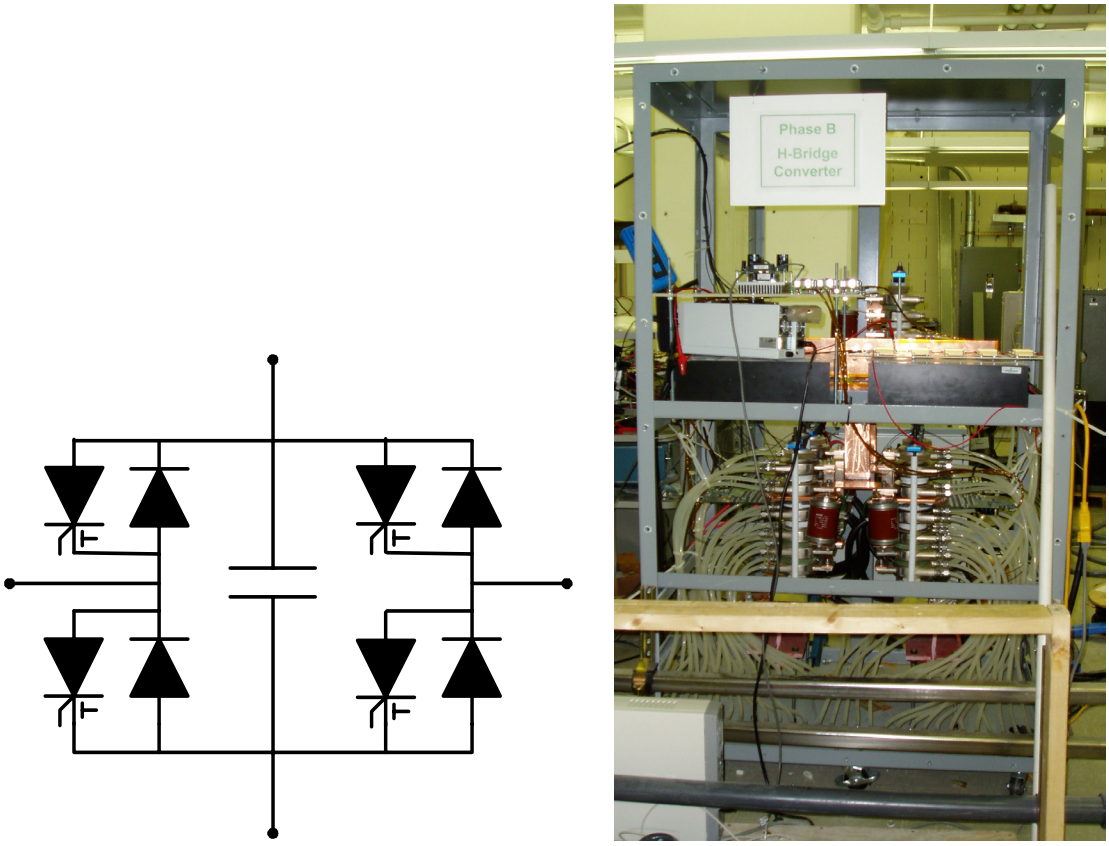


Figure 61: ETO Based HBBB: Schematic (left), Physical (right)

In addition to what is shown are precharge, discharge, and protection circuitry that are controlled via contactors and local circuitry.

The hardware manager should be able to support the HBBB as well as the totem-pole structure that it was originally intended for.

Table 6: Digital Signal Interface Requirements

PEBB PnP Phase Leg	TVA/DOE ETO based HBBB
1 Intelligent Power Module (IPM) with • 2 Command Signals • 2 Fault Signals A line contactor with • 1 Command Signal • 1 Status Signal	4 Emitter Turn off Thyristors each with • 1 Command Signal • 1 Status Signal • 1 Fault Signal • 1 PWM Current Measurement A precharge contactor with • 1 Command Signal • 1 Status Signal A discharge contactor with • 1 Command Signal • 1 Status Signal A line contactor with • 1 Command Signal • 1 Status Signal Status from a crowbar circuit
Maximum of 6 Digital Interface Signals	Maximum of 23 Digital Interface Signals

4.2 Hardware Manager Architecture

The hardware manager has an FPGA that is central to its operation. Unlike the Universal Controller's bus style, all peripherals branch from the hardware manager on separate lines, eliminating the busses, as shown in Figure 62.

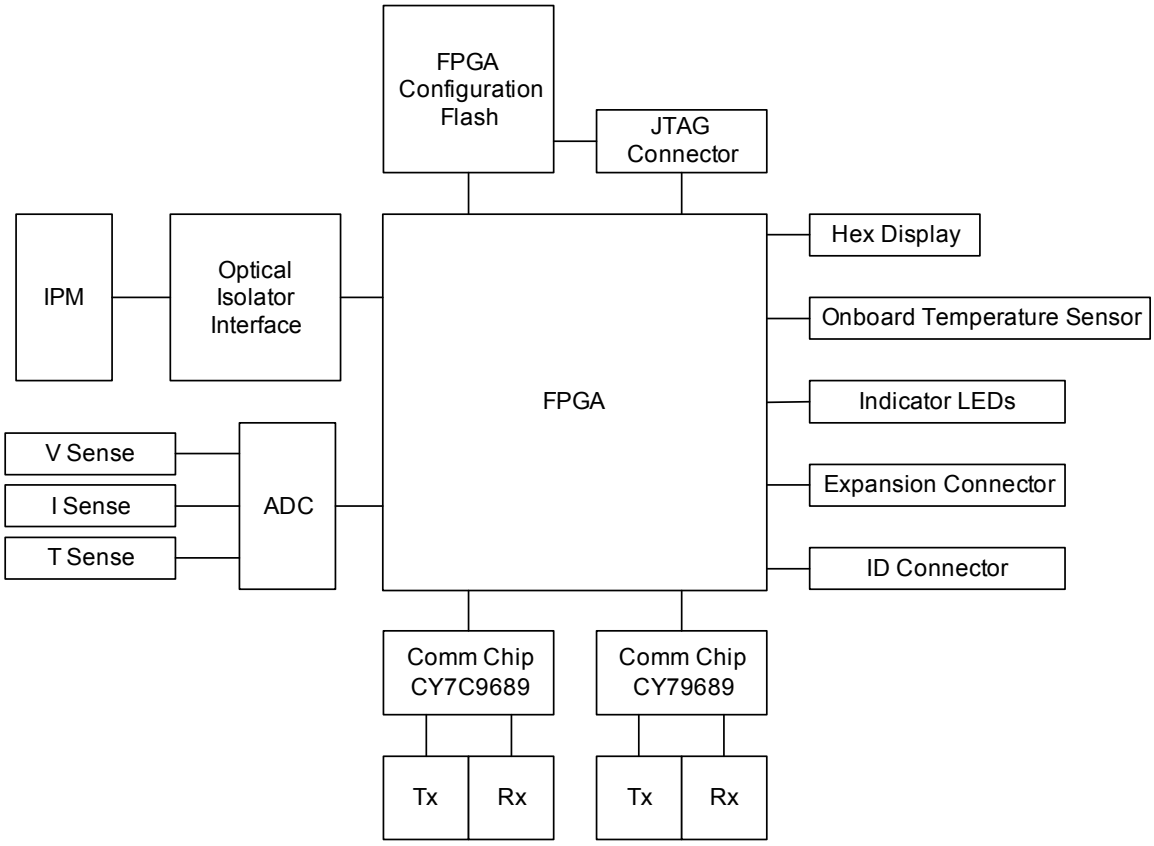


Figure 62: Hardware Manager Block Diagram

The FPGA chosen was a Xilinx Virtex-300. This is a large FPGA, and will be large enough to support future research and future versions of the PESNet protocol. Originally, the Virtex-E series was going to be used, but it was later changed because the I/O pins were not 5 Volt tolerant, which caused problems for the 5 Volt Cypress Communication chips. Solutions to the 5-Volt tolerance issues were to add series resistors to each line, or to add buffers between the FPGA and the device. Neither of these solutions were feasible due to the high density and the order of the signals on the PCB. The routing would have become incredibly more complex, as the direction of each I/O line now becomes important due to the buffer chip layout, more space taken by buffers and their bypass capacitors, and the high density of the buffers necessary. There are 96 lines between the FPGA and the communication chips alone. Each of these lines would have to be routed first to or from a buffer. The high density of these buffers would make them hard to populate, as they were high density BGA (ball grid array) components

(0.8mm pitch) that would require that the PCB trace width be reduced to 6 mils instead of the 9 that we were using, increasing the cost of the PCB. There was also a possibility that the number of blind via layers would have increased which significantly increases the price of the PCB, and adds other artifacts such as increased warp and twist.

The star like configuration allows the hardware manager to be debugged incrementally, disabling features until they are desired. It also allows the VHDL code to be simpler, and eliminates the possibility of bus contention, as all of the data, address, and control lines are unidirectional on the hardware manager.

A picture of the hardware manager is shown in Figure 63.

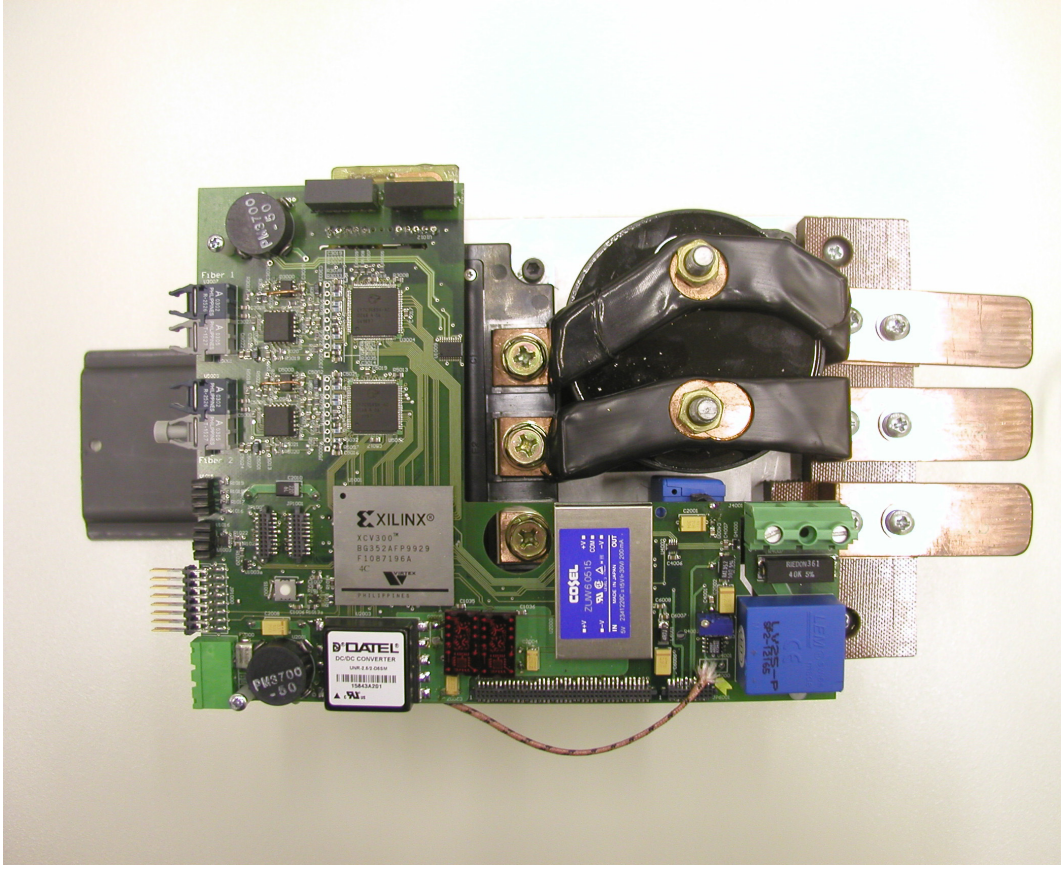


Figure 63: Hardware Manager

4.3 FPGA Architecture of Hardware Manager

Due to the star layout of the hardware manager, the FPGA VHDL code is similar to the physical layout and routing. Each peripheral has a controlling block. For some blocks, there are interfering control inputs that come from a limited number of other blocks, such as a protection block. The list of blocks and their functions are shown in Table 7.

Table 7: FPGA Blocks for Hardware Manager

Block	Function
HMTOP	Top level interconnect
PESNet	Communication protocol support
ADC	Analog to Digital Converter control
PWM	PWM Generator
Protect	Protection control and indication
Hex	Hex display control
HMSYSTEM	Controls miscellaneous hardware manager functions
TEMPSENSE	Controls the onboard temperature sensor

During the initial tests of the hardware manager, a high current was needed to see if the EMI affected it, but a high power was not required. At the early stage of development, it is also preferable to avoid high current under uncertain conditions. In order to test the hardware manager under high current operation, a two-pulse test was run. Due to the modular nature of the hardware manager VHDL and hardware architecture, it was easy to exchange the PWM block with a two-pulse test block.

5 PESNet Communication Protocol

Each node in the system has to communicate with each other node. One major advantage that this system has over most fieldbus protocols and other higher level protocols is its synchronous nature. PESNet ties each, otherwise isolated node together on a common fault tolerant communications bus.

5.1 Overview of PESNet 1.2

PESNet 1.2 was originally implemented in [21]. This version of PESNet can be characterized by the following features:

- Uses Plastic Optical Fiber (1mm) at 650 nm
- 125 Mbps
- 4B/5B encoding
- NRZI encoding
- Daisy chained ring architecture

In a daisy chained ring, each node has only one transmitter and one receiver. The transmitter of one node is directly connected to the receiver of another node to form a ring, as shown in Figure 64

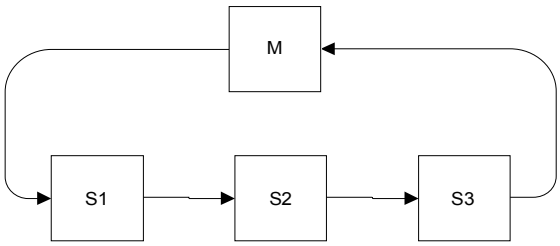


Figure 64: Single Ring Daisy-Chain Architecture

The physical layer was implemented using AMD 7968 and AMD 7969 transmitters and receivers. The AMD chips performed the 4B/5B encoding and decoding. In 4B/5B encoding, 4-bit characters are re-encoded to become 5-bit characters. This helps with balancing capacitors in the transmitters and receivers.

When encoding a character using the AMD chips, it is possible to have either a command character or a data character. These two are encoded differently. The following table shows the conversion from 4-bit to 5-bit data characters. Since 1 byte is transmitted at a time, two characters are concatenated together to form a complete word.

Table 8: 4B/5B Data Encoding

HEX Character	Binary Representation	Encoded 5-bit data character
0	0000	11110
1	0001	01001
2	0010	10100
3	0011	10101
4	0100	01010
5	0101	01011
6	0110	01110
7	0111	01111
8	1000	10010
9	1001	10011
A	1010	10110
B	1011	10111
C	1100	11010
D	1101	11011
E	1110	11100
F	1111	11101

If the character is a command, the following table shows the encoded command character. Unlike a data word which is 8 bits of unencoded data, a command word is 4 bits, and thus there are only 16 possible commands.

Table 9: 4B/5B Command Encoding

Command Name	HEX Command	Binary Representation	Encoded 5-bit command character
JK	0	0000	11000 10001
II	1	0001	11111 11111
TT	2	0010	01101 01101
TS	3	0011	01101 11001
IH	4	0100	11111 00100
TR	5	0101	01101 00111
SR	6	0110	11001 00111
SS	7	0111	11001 11001
HH	8	1000	00100 00100
HI	9	1001	00100 11111
HQ	A	1010	00100 00000
RR	B	1011	00111 00111
RS	C	1100	00111 11001
QH	D	1101	00000 00100
QI	E	1110	00000 11111
QQ	F	1111	00000 00000

Command 0 (JK) is reserved as a synchronization character. When the transceiver receives this pattern, it resynchronizes its character clock (and therefore the character boundary) with the start of this pattern. It is important for the character clock to be synchronized before it recovers valid data from the serial stream. From the pattern below, it is not possible to determine the start of a byte without some pattern to "trigger" on.

…01111100100111000100011010101010111100100 1…

Once a pattern has been defined, the character boundary can be fixed, and it is possible to start extracting characters:

…01111100100111000100011010101010111100100 1…

…????????,JK,34,01

The data is encoded using NRZ encoding. In this encoding, a transition from '1' to '0' or from '0' to '1' is used to represent a logical '1', and no transition is used to represent a '0'. This is shown in Figure 65.

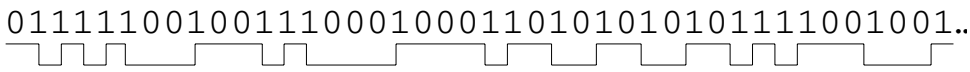


Figure 65: NRZ Encoding

From this stream, it is possible to recover both the clock using a phase-locked loop and the data. Up to now, these functions have been provided by the AMD TAXI chips. The TAXI chips generate a PECL signal at the output and read a PECL signal at the input of the analog side of the chip. These signals are differential. They go into an electrical to optical transmitter and receiver circuit designed by Agilent Technologies [67]. The output of the transmitter goes to the fiber, where it is recovered at the next node.

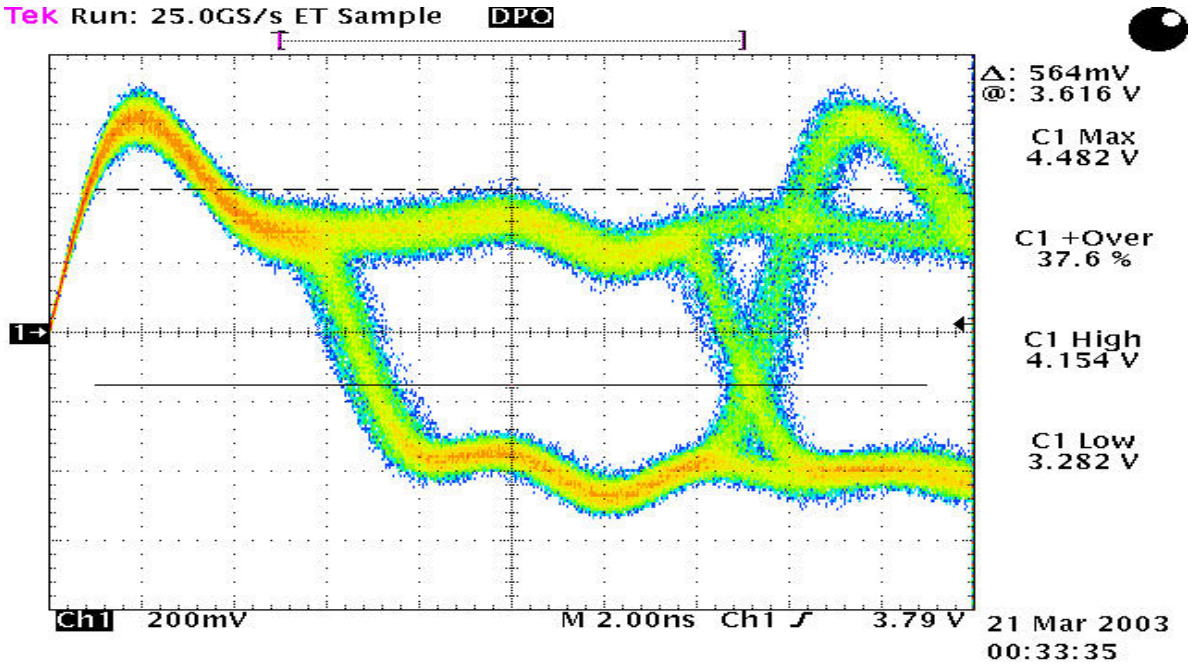


Figure 66: PECL Signal At Optical to Electrical Interface

The characters sent into the digital side of the communications chip represent part of a packet of data. There are two types of packets sent in PESNet 1.1. The first is a data packet, which is designed to deliver information to a specific slave device on the network. The master initiates all packets. The format of the data packet is shown in Figure 67.

	ADDR	BYTE1	BYTE2	BYTE3	BYTE4	CRC
Size	4 Bits	8 Bits	8 Bits	8 Bits	8 Bits	8 Bits

Figure 67: PESNet 1.2 Data Packet Structure

The first field in the packet is the address of the node to send the data to. The following four fields are the different bytes of data. Typically, these would contain information such as the duty cycle. When a slave receives a piece of data that is not destined for it, it will forward the information to the next node. This happens on a byte by byte basis. That is, the EPLD on each node stores only one byte of transmission data at a time. At a given time, different parts of the same packet may be in three nodes simultaneously.

Each slave receives data from the master, and returns sensor data back to it in a packet with the same ADDR field. The master collects this data, and uses it for closed loop control.

In power electronics, it is important to synchronize actions for all switches. This is accomplished with a second packet, called a synchronization packet. After information is sent to all three nodes, the synchronization packet is sent to activate the data. The synchronization packet has the following structure:

	CMD	ADDR3	PADDING	ADDR2	PADDING	ADDR1
Size	4 Bits	8 Bits	8 Bits	8 Bits	8 Bits	8 Bits

Figure 68: PESNet 1.2 Synchronization Packet Structure

The concept is that each node will synchronize when they receive their own address in the data field. Since the addresses are sent in reverse order, all nodes should receive their own address at exactly the same time, synchronizing the nodes.

5.2 PESNet 1.2 Drawbacks

Given the methodology used to implement some of the desired features, it gives rise to some restrictions. The most restrictive is the dependency on node order in the ring. If an

extra node is added in the ring between any other nodes, the extra transmission delay caused by adding an extra node would desynchronize the nodes due to the method used to synchronize them.

Each node on the ring is a single point of failures. If any node fails, the data path from the master to the slaves and back to the master is interrupted, breaking the "closed loop".

The data is assigned to functions by its position within a data packet only. Lack of variety in the data packet reduces the usability of PESNet 1.1. For example, the duty cycle is always located in position 1 and 2 of the data packet. If more than 4 bytes are needed to completely control the node, the node would have to have two addresses.

The number of nodes is limited to the number of commands. If more nodes are needed, it would be impossible to add them.

Some applications require more data and less timing constraints. In these cases it is desirable to have a packet that is large. Other applications require less data and faster timing. It is difficult to meet both of these application's requirements without having the packet size be adjustable.

There is also a problem if there different requirements for synchronizing different nodes. If one group of nodes had to be synchronized every 100 microseconds, and another group of nodes had to be synchronized every 70 microseconds, then it will become complicated to synchronize these nodes, especially every 7 milliseconds, for example, when the synchronization times are nearly the same, and two packets have to be on the network at the same time, possibly at the same nodes (which would be impossible to realize). Moreover, the order of the packets would have to vary.

5.3 Overview of PESNet 2.2

While many features are kept the same as in PESNet 1.2, PESNet 2.2 was created to address some of the issues introduced by 1.2. Some of the improvements are listed below.

- Improved synchronization
- Multi-master system
- Dual ring, fault tolerant structure
- Fault tolerant clock scheme
- Multiple data priorities
- Management of synchronous and asynchronous data
- Adjustable packet size
- Hot-swapping support
- Live insertion and removal of nodes
- Variable ring structure – relieves dependency on node order
- Support for multiple groups of synchronized nodes at different frequencies
- Self-healing

One of the most significant features changed is the synchronization scheme. It is easy to explain the synchronous nature of PESNet 2.2 as a rotating gear with n teeth rotating in a circular cavity with n slots, as shown in Figure 69, where n is shown as 8. Between every two adjacent teeth is a packet of information. The gear rotates in a lock-step fashion, moving to the next of the n positions and locking. While locked, the packet is read and is either replaced by another packet of information, or is consumed (effectively replacing the original ball with a "NULL" packet.), or remains untouched. The gear will continue to rotate in this fashion, where at all times, all nodes have a packet to replace or ignore. All packets leave the current node and arrive at the next node at the same time. In the middle of the gear is a counter (clock) that is incremented every time the gear rotates to the next notch. Every node has the same value of this clock.

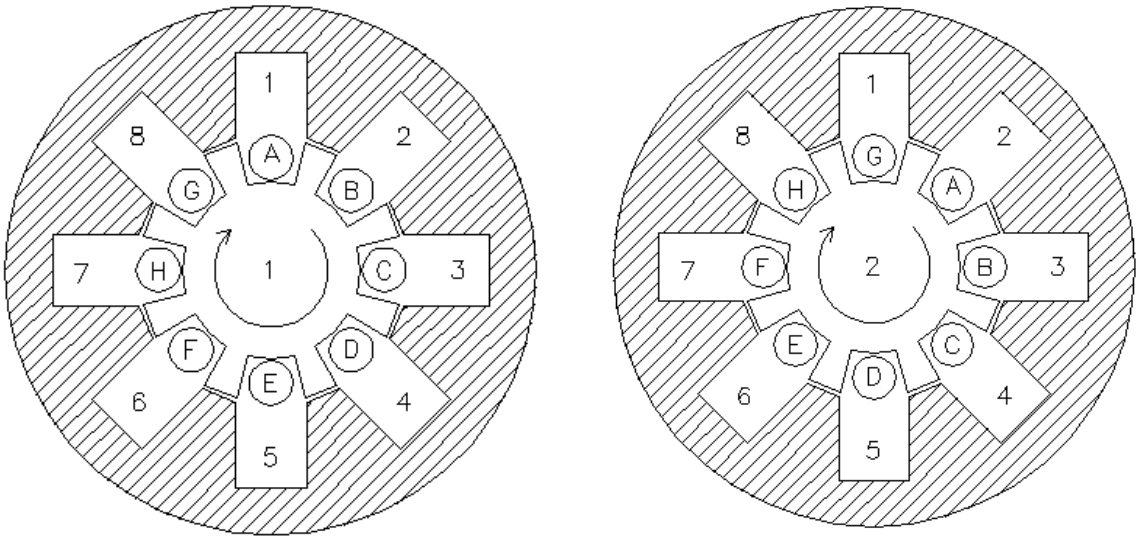


Figure 69: Gear Conceptualization of PESNet

While the actual timing varies from this representation, it is effective to describe the timing of PESNet like this, as the instant that a full packet is received by one node is the same instant that the full packet is received at all nodes. In actuality, each node starts transmitting data at the same time on a byte-by-byte basis. When node 4 is transmitting its first byte, so is node 2, and all the other nodes as well. As the data enters the transceiver chips, the data propagates through the FPGA, where it is processed as it arrives, setting up internal conditions in the FPGA to commit the command or send sensor data. When the last real byte is sent, and the packet is determined valid, the action is committed, and the now ready data flows directly to the transmit buffer of the transceiver. In the mean time, two JK characters are being sent and received as "padding" on the end of the packet. In addition to giving the FPGA the necessary 80ns to finish reacting to the new packet, the JK character also serves the purpose of realigning the decoder in the Cypress chip to prevent errors. At all times, each nodes must transmit a packet, even if it is an empty (NULL) one.

5.4 Packet Structure

The PESNet packet has seven fields. A typical PESNet packet is shown in Table 10. One benefit of PESNet 2.2 is that the packet size can be set at the beginning of operation. One would use a smaller packet if there were more nodes that need to be controlled, and

it is acceptable to use less information to control each node. Otherwise, one would use a larger packet size if there were fewer nodes. The reduced packet size allows more packets to be sent and received within a shorter amount of time, which is useful for higher switching frequencies. The tradeoff is the amount of data sent to each device, as well as the maximum size of the addresses, and network clock.

Table 10: Packet Size and Breakdown by Mode

	SYNC	CMD*	DEST ADDR	SRC ADDR	NET TIME	FAULT ADDR	DATA ARRAY	CRC
Full Mode	8 Bits	4 Bits	8 Bits	8 Bits	8 Bits	8 Bits	10 Bytes	8 Bits
Reduced Mode	8 Bits	4 Bits	6 Bits	6Bits	6 Bits	6 Bits	4 Bytes	8 Bits

* The total size is the size of the useful data. There are two more bytes sent of JK characters used for processing data
** The 4 Bit command is still encoded as though it was 8 bits

Under the full mode, the size of the packet is 16 bytes. Under the reduced mode, the size of the packet has been reduced to 9 bytes.

The individual fields are described in Table 11.

Table 11: Packet Field Descriptions

Field	Description
SYNC	Two JK Characters used to restore byte alignment if it has been lost prior to receiving packet
CMD	The command describes the packet type. The nature of a command byte is such that it also indicates the beginning of a packet, as it is encoded differently that normal data
DEST ADDR	The node that is the destination of the data. Future PESNet versions will allow groups of nodes to have addresses.
SRC ADDR	The address of the node that sent the packet
NET TIME	The current value of the network clock
FAULT ADDR	The address of a faulted node. Zero if no node is faulted.
DATA ARRAY	The data that is delivered to the node. The format of this array depends on the command.
CRC	A cyclic redundancy check to ensure that the packet has not been corrupted during transmission.

5.5 Types of Data

In order to determine what data the node should support, it is important to look at the applications that this version of PESNet should support. While PESNet 1.1 was focused primarily on the steady state operation of a single converter, PESNet 2.2 extends this to consider configuration, startup, failures, shutdown, and reconfiguration.

There are events that occur during these other states that must be supported, some of which also have a requirement to be synchronized. Other objectives, such as plug and play support require information to be set and read by the slaves for configuration purposes. This information has no relationship to time. Examples of this information would be a manufacturer ID that would describe who manufactured the PEBB, or a revision number. There are always unexpected events, such as failures, limit crossings, or emergency shutdowns. All of these types of data must be handled by the protocol.

PESNet v 2.2 has four types of information that can be transmitted on the network. Corresponding commands indicate the type of data that is represented in the data array within the packet. These data types are listed below:

- Normal Data
- Synchronous Data
- Asynchronous Data
- Events

Normal Data. Normal data is data that is exchanged every switching cycle. This information is typically a duty cycle for a phase leg. Normal data was the only data type available in PESNet 1.1. This is used for realtime control, and watchdog timers can be set to fire if the node does not receive normal data from another node in a certain time frame.

A differentiating factor between normal data in PESNet 2.2 and PESNet 1.1 is the methodology for synchronizing nodes. Here, the network clock is used. Normal data has a time associated with the data that it will become active. The majority of packets on PESNet will consist of normal data, as this is the data typically used for synchronous realtime control.

Synchronous Data. Synchronous data is data that becomes active at a specific time, similar to normal data, but is not expected to occur at every switching cycle. Synchronous data may be a command for the state of a contactor. For all nodes, the contactors should open at the same time. In order to synchronize this "random" event with other contactors, a time that this data is valid is given. The data will not change until the network time is the time specified by the packet. A watchdog timer is not normally used for this data, as this data is not expected within a certain time frame.

Asynchronous Data. Asynchronous data is data that does not depend on the value of the network clock. As soon as a packet arrives at the node with asynchronous data, that data immediately becomes active. The data is called asynchronous because the time when the

data becomes valid cannot be synchronized with any other node. This asynchronous data is characteristic of most industrial fieldbusses.

Events. Events are special data in that they can have implications at the protocol level. While the other three data types provide information to the application layer of the protocol, whether the application is a hardware manager or a universal controller, some smart sensor or smart passive, an event can cause control variables in the lower levels of the protocol to change. Some events are described below:

- Modify network clock time
- Inject fault in network clock
- Inject fault into node (emulate node failure)
- Emergency shutdown all (this is a network level, because it will immediately forward this packet to other nodes in both directions, overwriting data that is going to be sent)
- Set address
- Node fault

The structure of a slave's memory is shown in Table 12. To differentiate between the address of a node and the "address" in memory of a specific data value, the memory is called attributes, and attributes are assigned attribute numbers ("address" of attribute).

Slave nodes will have the following memory map:

Table 12: PESNet Data Types

Data Type	Memory Size
Normal Data	4 x 2-Byte Attributes
Synchronous Data	32 x 2-Byte Attributes
Asynchronous Data	32 x 2-Byte Attributes
Event Data	32 x 1-bit events

5.6 Commands

Table 13 describes the possible commands used in PESNet 2.2.

Table 13: PESNet Commands

Pattern	Command Name	Description
0001	NS_NORMAL	Set the normal data and reply with the normal sensor data
0101	NS_GET_SYNC	Get a synchronous data from the synchronous data manager
0100	NS_SET_SYNC	Set the synchronous data for the synchronous data manager
0010	NS_GET_ASYNC	Get asynchronous attribute value from the node
0011	NS_SET_ASYNC	Set a node's asynchronous attribute
1000	NS_EVENT	Specifies an event sent from another node
0110	NS_NULL	Specifies an empty packet
1001	NS_REQ_PARAM	Get parameters from neighboring node
1010	NS_SET_PARAM	Set network parameters from neighboring node
0111	NS_COMMAND	Send and process network commands

NS_NORMAL. This command transmits normal data in the packet. The format of the data payload is as shown in Figure 70. The first field, DATA[0], contains the direction of the data in the most significant bit. A 0 indicates that the data is being sent from the master to the slave, and a 1 indicates the slave's reply. The key is a transaction number for reference by the master if desired. During a transaction, the slave node receives a packet with a 0 in the direction field, and a key. When the slave replies to the packet, the transaction number is kept the same, but the direction is set to '1', indicating that this is a reply with sensor data. Since masters can also be slaves, this is necessary to indicate that the packet is a response, and not a command from that node to the master. The key is kept the same in the reply as it was in the original packet. The reduced mode allows one 16-bit data normal data to be set.

The Time field is the time at which the data can become active. For PESNet 2.2, there can only be one pending NORMAL data packet. Succeeding ones will replace the previously pending one. For each device, the meaning of the data are predefined, depending on their device type.

	0	1	2 3	4 5	6 7	8 9
Full Mode	DIR&KEY	TIME	DATA1	DATA2	DATA3	DATA4
Reduced Mode	DIR&KEY	TIME	DATA1			

Figure 70: NS_NORMAL DATA PAYLOAD

NS_GET_SYNC. This packet requests synchronous data from the slave in a reply packet. The format of the packet data payload is shown in Figure 71. The ATTRx fields are the attribute numbers for the data being requested. The DIR field is again the most significant bit, and has the same meaning as before. The DATA fields have no meaning at the time of request, and are filled in by the slave, and the new packet is sent back. The slave only changes the DIR field and the DATAx fields. The rest of the packet (ATTRx and KEY) remains untouched.

	0	1	2 3	4	5 6	7	8 9
Full Mode	DIR&KEY	ATTR1	DATA1	ATTR2	DATA2	ATTR3	DATA3
Reduced Mode	DIR&KEY	ATTR1	DATA1				

Figure 71: NS_GET_SYNC DATA PAYLOAD

NS_SET_SYNC. This packet sets synchronous data in the slave. The synchronous data is time-activated, and thus every corresponding data has an activation time. This packet does not warrant a reply, and thus there is no need for a direction parameter or a key. The format is shown in Figure 72.

	0	1	2	3	4	5	6	7	8	9
Full Mode	TIME1	ATTR1	VALUE1		TIME2	ATTR2	VALUE2		RESERVED	
Reduced Mode	TIME1	ATTR1	VALUE1							

Figure 72: NS_SET_SYNC DATA PAYLOAD

NS_GET_ASYNC. This packet requests asynchronous data from the master. Asynchronous data does not depend on time, and thus there are no fields in the data payload relating to the network clock. The DATAx fields are set in the reply packet.

	0	1	2	3	4	5	6	7	8	9
Full Mode	DIR&KEY	ATTR1	DATA1		ATTR2	DATA2		ATTR3	DATA3	
Reduced Mode	DIR&KEY	ATTR1	DATA1							

Figure 73: NS_GET_ASYNC DATA PAYLOAD

NS_SET_ASYNC. This packet sets the value of asynchronous data.

	0	1	2	3	4	5	6	7	8	9
Full Mode	RESERVED	ATTR1	DATA1		ATTR2	DATA2		ATTR3	DATA3	
Reduced Mode	RESERVED	ATTR1	DATA1							

Figure 74: NS_SET_ASYNC DATA PAYLOAD

NS_EVENT. The Event packet has a key and a time of transmission. It can originate from any node.

	0	1	2	3	4	5	6	7	8	9
Full Mode	KEY	TIME	DATA1							
Reduced Mode										

Figure 75: NS_EVENT DATA PAYLOAD

NS_NULL. The NS_NULL packet is an empty packet with no meaning. It provides a space for a new packet to be placed on the network.

	0	1	2	3	4	5	6	7	8	9
Full Mode	Zeros									
Reduced Mode	Zeros									

Figure 76: NS_NULL DATA PAYLOAD

NS_REQ_PARAM. This packet requests the network parameters from the neighboring nodes. This packet consists of only the command field, and has no payload. The neighboring node will answer the request using a NS_SET_PARAM packet of the correct size with the correct value of the network clock, as if it were regularly communicating with this node.

NS_SET_PARAM. NS_SET_PARAM packets set the network parameters to enable a node to communicate. This packet is sent in response to a NS_REQ_PARAM packet.

	0	1	2	3	4	5	6	7	8	9
Full Mode	nVersion	nSize								[JF1]
Reduced Mode	nVersion	nSize								

Figure 77: NS_SET_PARAM DATA PAYLOAD

NS_COMMAND. Use a series of extended commands for the network, such as ping and set address. For commands which have a reply, the direction field is available, as well as a key.

	0	1	2	3	4	5	6	7	8	9
Full Mode	DIR&KEY	COMMAND	DATA1		DATA2		DATA3		DATA4	
Reduced Mode	DIR&KEY	COMMAND	DATA1							

Figure 78: NS_COMMAND DATA PAYLOAD

5.7 Synchronization and Net Clock

As described before, the network is synchronized using a network clock. The network clock is incremented with every packet transmitted. Ideally, the value of the network clock is the same for all nodes. If there is a condition where the network clock is not valid, there is some fault tolerance present to ensure that it can be corrected. There is a module in the PESNet protocol stack called the network clock manager. This manager is responsible for synchronizing the network clock. At any time, there are three values of the possible network clock. These values create a triply redundant set. These values are:

1. The value in the packet received from the left
2. The value in the packet received from the right
3. The previous value of the network clock + 1

In addition, any packet can be ruled out if the CRC did not pass for it.
During initialization, the network clock is set using a NS_COMMAND packet. This packet forces the value of the network clock. Once this packet travels around the ring, the same node that generated it absorbs the packet. At this point, all nodes should have the same value of the network clock.

The network clock is always valid, even if the packet is a null packet. Once the network clock is synchronized, all events depending on the value of the network clock will also then be synchronized.

5.8 Dual Ring Fault Tolerance

The use of two rings adds a fault tolerant property to the network. SONET uses two rings in the same way, referred to as a unidirectional path switched ring (UPSR) [74]. In the event of a link or node failure, it is possible to reverse the direction using the second ring. During unfaulted operation, the outer ring would be used to transmit data and control, as shown in Figure 79. The inner ring would transmit null packets.

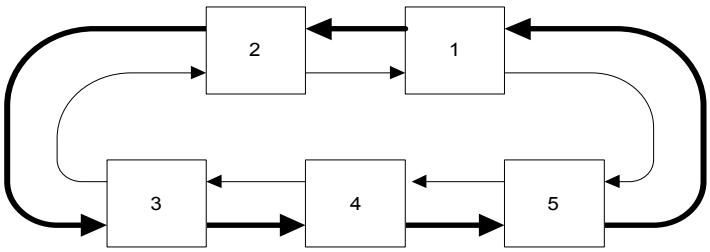


Figure 79: Normal Dual Ring Operation

If one of the nodes were to fail, then the data transmitted to that node previously would be sent using the inner ring to the previous node. The data being received would go into an internal buffer. This configuration is shown in Figure 80. In this version of PESNet, data would still be processed on the primary ring for simplicity, however. In the case of the end node, the buffer would be used.

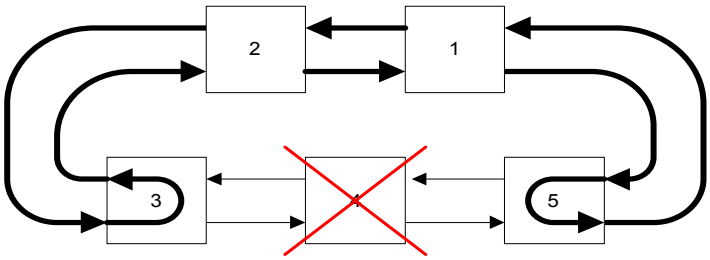


Figure 80: Faulted Dual Ring Operation

The dual ring architecture is managed by a block in the protocol stack called the redirector. This block is responsible for transmitting the data to the correct "port". The redirector has the capability to change the flow of packets, as well as to heal the network after the fault has been fixed. The structure of the redirector is shown in Figure 81.

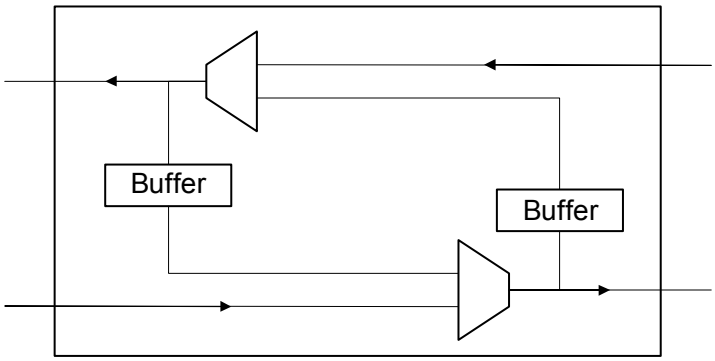


Figure 81: Redirector

5.9 TAXI Chip and Physical Layer

The Cypress TAXI compatible HotLink transceiver [75] is a replacement for the original AMD 7968 and 7969 TAXI transceivers. The new chip is a 100-pin TQFP device with additional features beyond that of the original 28 pin ceramic DIP transmitter and receiver.

The optical interface is at the top of the PCB. The current transceiver communicates using 1mm plastic optical fiber (POF) transmitting 650nm light. At the bottom of the figure are the Cypress chips.

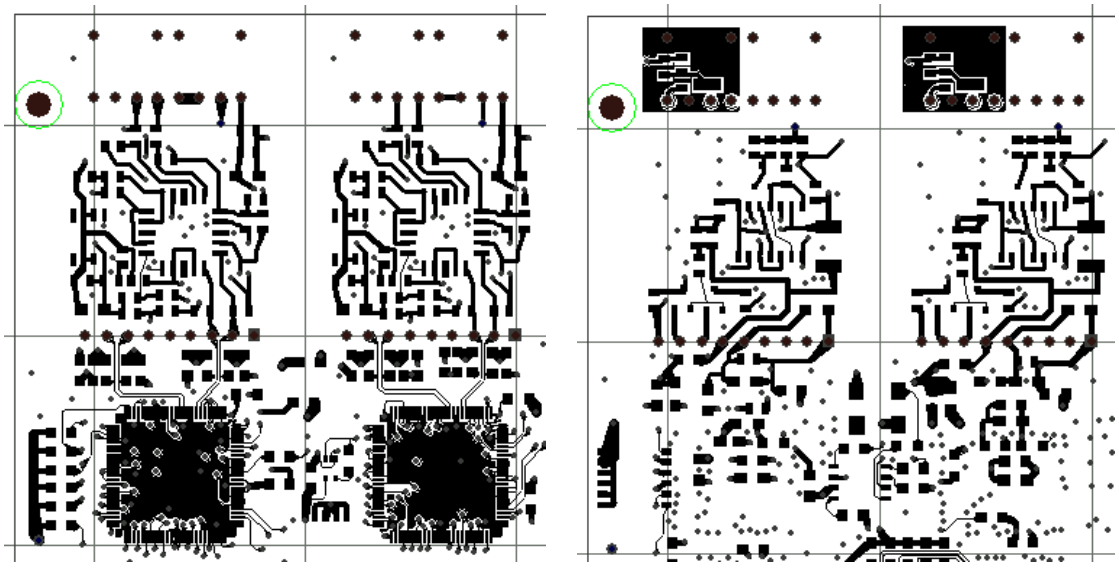


Figure 82: TAXI Chips, Optical Transmitters and Receivers

The layout for the optical transmitter and receiver is specified by Agilent technologies in application note 1066 [67]. The input to the optical transceivers is a PECL (pseudo emitter-coupled logic) signal that is sent and received by the Cypress chips.
The Cypress chip provides several functions. This design makes use of the following specific ones:

- Serializing/Deseralizing Data
- NRZI encoding and decoding
- 4B/5B encoding and decoding
- Link health
- Encoding error detection
- Automatic alignment using JK characters
- Distinguishing between command and data characters

5.9.1 LLI stack

PESNet follows the OSI model, and has four layers: Application, Network, Datalink, and Physical. The lower three layers are common to any PESNet node, regardless of its function. Collectively, these layers are referred to as the lower level interface, or LLI. These layers are responsible for:

- Composing/decomposing the packet
- Checking the integrity of the packet
- Managing the network clock
- Filtering commands
- Redirecting packets during faults
- Adding and removing packets from the network and replacing them with NULL packets if necessary
- Implementing attributes
- Implementing simple DSP interface

The LLI sits between the application and the Cypress Transmitter. It is completely implemented in the FPGA. The primary LLI PESNet interface operates at 80 MHz. At

the highest level in the LLI, there are five managers. Each of these managers handles a different type of command on PESNet. The managers are described below.

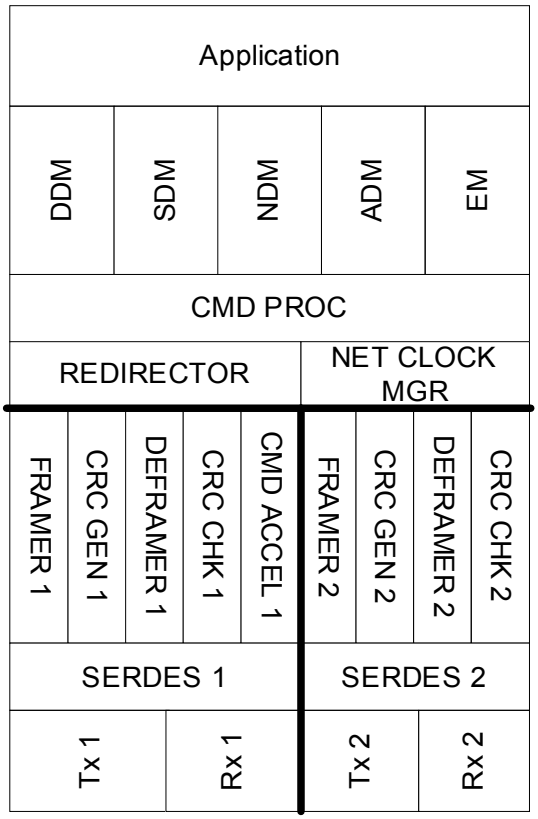


Figure 83: PESNET LLI Stack

Asynchronous Data Manager (ADM). The asynchronous data manager handles NS_SET_ASYNC and NS_GET_ASYNC commands. This data typically contains the manufacturer and vendor information for the node, local DIP switch settings, HEX values, and other data that is not time critical, such as protection thresholds set prior to converter operation, and even water cooling information.

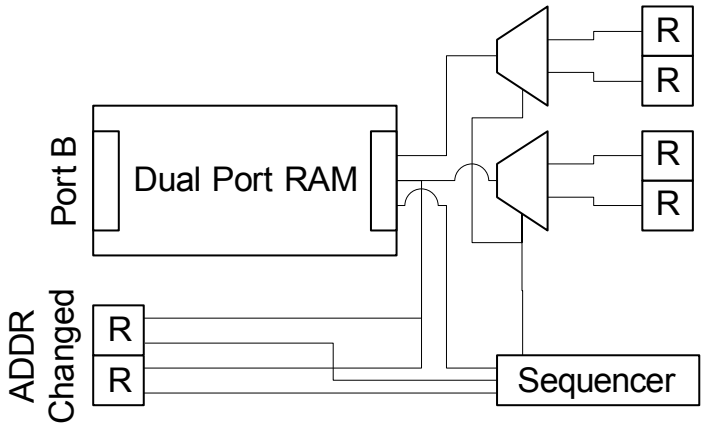


Figure 84: Asynchronous Data Manager

Event Manager (EM). This manager handles events generated at the local node, as well as those passed over PESNet via the NS_EVENT command. Typical NS_EVENT messages are local node failures, warning thresholds, and user events. Typical received NS_EVENT messages are shutdown commands, faulty link warnings, network level interrupts, and fault injection commands for test.

Normal Data Manager (NDM). This manager handles data that is critical to the operation of PESNet. Data for this manager is set prior to the time it should be active. Along with this data is a network time that indicates that the data should be activated. When the time occurs, all of the data delivered in the normal data packet will be activated simultaneously via a double buffer. Examples of such data are PWM modulator information. The normal data manager always replies with sensor data. The sensor data replaces the normal data packet, and is sent back to the render of the original packet. Data is always loaded into the normal data manager from a single packet. Successive NS_NORMAL packets replace the pending data.

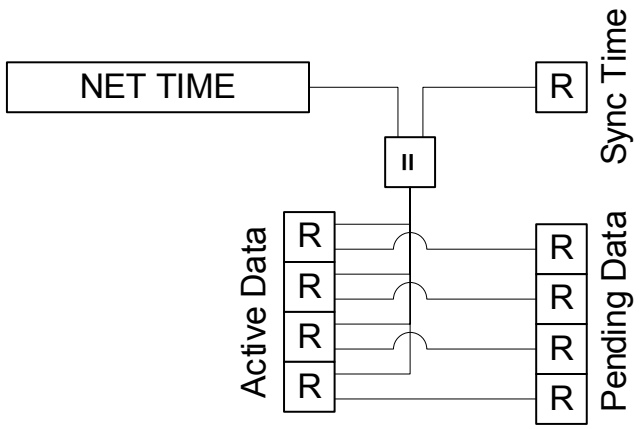


Figure 85: Normal Data Manager

The actual implementation is shown in Figure 86.

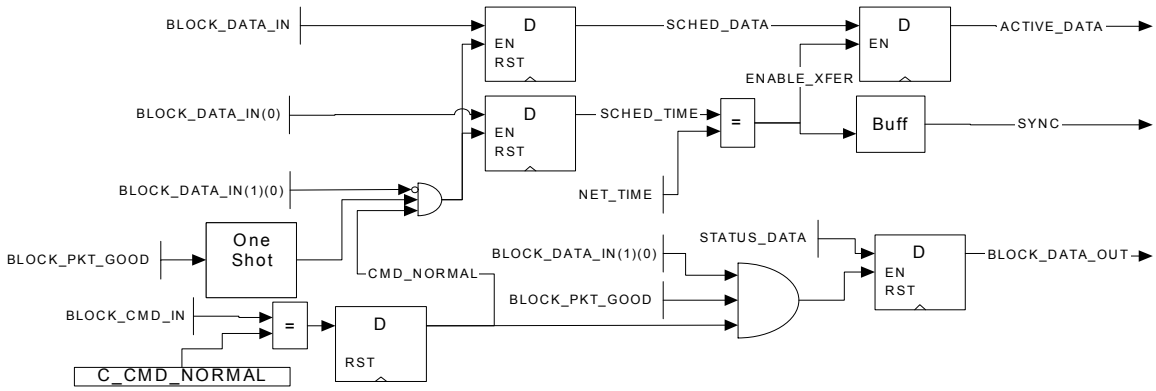


Figure 86: Normal Data Manager Implementation

This is coupled with a state machine to inform the next layer down (command processor) that there is a packet to be transmitted. The state machine for the normal data manager is shown in Figure 87.

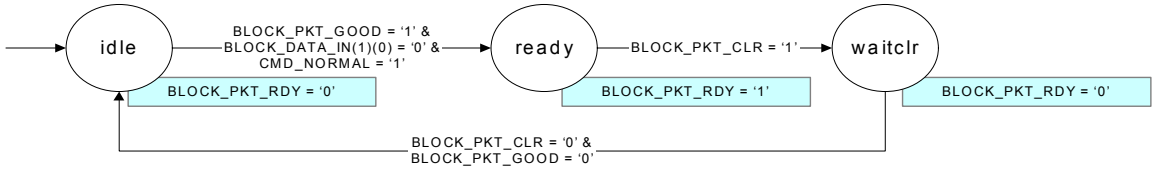


Figure 87: Normal Data Manager State Machine

Synchronous Data Manager (SDM). This manager contains an array of data that is time sensitive. Unlike the normal data manager, this manager does not reply to packets

after the data is set. Each piece of synchronous data comes with a time at which the data should be activated. Data typically controlled from this manager are things such as contact states, normal operation modes, and scheduled fault injection.

The synchronous data manager is implemented using a double buffer structure. There are three arrays of data. The first array is data that is valid at the time. All reads of synchronous data come from this array. The second array consists of pending data. The third array consists of the time at which the pending data will become valid. The two data buffers are cascaded together, with the time passing through a comparator into the load enable of the second buffer, as shown in Figure 88. Figure 69 only shows five registers, but there are actually 32 of them.

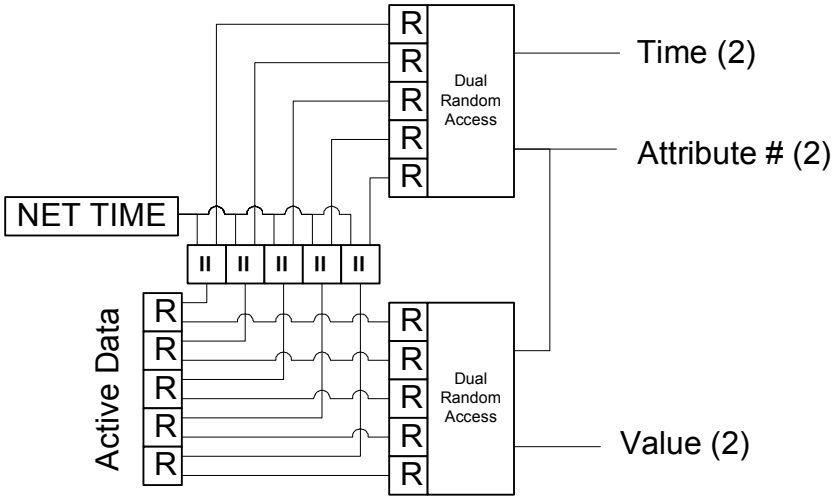


Figure 88: Synchronous Data Manager Structure

DSP Data Manager. For nodes with DSPs, an interface has been created that buffers data packets received from the DSP, located on the other side of the manager. These managers are configured to support future ONR projects, including smart passive PEBBS.

The DSP data manager has two pieces. One is simply a set of buffers with an address sensitivity mask. When a packet arrives, the address is compared with the mask, and if it

meets the criteria, the packet is captured, and made available to the DSP for processing. The second data is the normal data of the slaves that the DSP is controlling. It may not make sense to collect this data at every packet. Instead, the data is stored in a table that is accessible from both the DSP and the LLI.

Row	Address	DATA_IN	DATA_OUT	TIME_IN	TIME_OUT	AND 1	AND 2	AND 3	OR	NEW	ENO
1	11	xx xx xx xx	yy yy yy yy	20	50	1	0	0	0	0	1
2	3	xx xx xx xx	yy yy yy yy	20	50	1	0	0	0	0	1
3	21	xx xx xx xx	yy yy yy yy	20	50	1	0	0	0	0	1
4	16	xx xx xx xx	yy yy yy yy	24	70	0	1	0	0	0	1

Figure 89: DSP Manager Register Table

Each field is the following size:

Table 14: DSP Manager Configuration Registers

Field	Size	Source	Quantity	Description
Address	1 Byte	DSP	32	Address of nodes
DATA_IN	8 Bytes	LLI	32	Data received from other node
DATA_OUT	8 Bytes	DSP	32	Control data sent to other node
TIME_IN	1 Byte	LLI	32	Time data was sent from other node
TIME_OUT	1 Byte	DSP	32	Time control data will become active
AND1	32 bits	DSP	1	"Vertical" register to trigger when all new data mask contains the same thing as the mask.
AND2	32 bits	DSP	1	Similar to AND1
AND3	32 bits	DSP	1	Similar to AND2
OR	32 bits	DSP	1	Triggers interrupt if any of these addresses show up
NEW	32 bits	LLI	1	Set when new data arrives
ENO	32 bits	DSP	1	Enables reply data to be sent out
PEND	32 bits	LLI	1	When packet is sent, goes to 0

All of these registers are memory mapped and are accessible from the DSP.

The setup of the normal data for the slave consists of the following procedure:

1. Identify an unused table entry
2. Ensure that the corresponding bit in ENO is clear. If this is set, the data may be transmitted before it is ready.
3. Set the address field to the address of the corresponding slave node
4. Set the AND1, AND2, AND3, and OR fields' corresponding bits. When the OR bit is set, and a new packet arrives, an interrupt will occur. When the AND bit is set, and all packets identified by the ANDx register arrive, then an interrupt will be set. This is used to manage different groups.
5. At this point, the ENO field has been cleared. To set the first data to be sent, write to the DATA_OUT register, and to the TIME_OUT register, and then set the corresponding ENO bit. When this bit is set, a NS_NORMAL packet will be sent out on the next available empty slot.

When an interrupt arrives, the corresponding NEW bit will be set. The interrupt can be determined by the following equation:

INTRPT1 = (NEW or not AND1) = X"FFFFFFFF"

INTRPT2 = (NEW or not AND2) = X"FFFFFFFF"

INTRPT3 = (NEW or not AND3) = X"FFFFFFFF[JF2]"

When ENO is written, it copies memory from the other side of the dual port ram in which the data table is to a second copy, where it waits for the DDM state machine to request the new data. Once the state machine requests the data, it is copied to a third tier of dual port ram. When it is finished copying, the state machine will inspect each row and transmit the necessary packets.

A second way to transmit a packet is using the special buffer. In this buffer, any packet can be sent from the DDM, including packets that are not NS_NORMAL. This is implemented as part of the same state machine that is used to manage the table based transmission method.

The state machine for transmitting packets is shown in Figure 90.

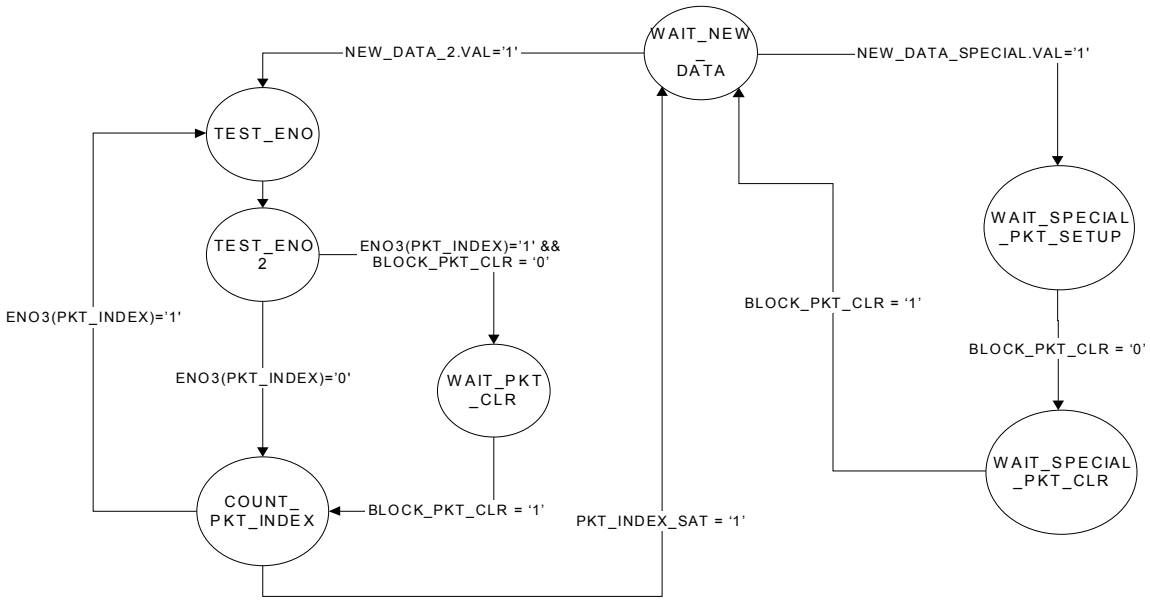


Figure 90: DDM State Machine

Here, WAIT_NEW_DATA is the idle state. When the data is finished copying to the third tier of the dualport memory, NEW_DATA_2.VAL will become 1. The state machine will then place the index of the current table row (starting from 0) in the dual port read address while in TEST_ENO. It will then evaluate the third copy of the ENO register (it transfers with each tier of the memory) to determine if there is a packet in that row that will be sent out. If not, it will skip the transmission state and increment the counter, PKT_INDEX, to the next value. If the counter has passed through all the possible rows, PKT_INDEX_SAT will become 1, and the state machine will return to the WAIT_NEW_DATA state, otherwise, it will go back to evaluating the next ENO bit. Transmission begins by seeing that the command processor is ready for another request (BLOCK_PKT_CLR = '0'). At this time, the data is presented to the command processor, and BLOCK_PKT_RDY is set to 1. When the command processor sends the packet, BLOCK_PKT_CLR is set to 1, indicating to the manager that it should set BLOCK_PKT_RDY back to zero to complete the transaction.

There are two ways to receive data in the DSP data manager. The first was the table based method described earlier. The second method is a FIFO based method that uses a circular array (shown in Figure 91) implemented in the FPGA to capture incoming

packets. A status register makes available the current amount of data in the FIFO. Packets that are not normal packets are captured here, and can be extracted via the DSP data manager interface. A FIFO increment flag is made available so that once the top packet has been read, it can be removed. The FIFO is implemented using dual-port memory.

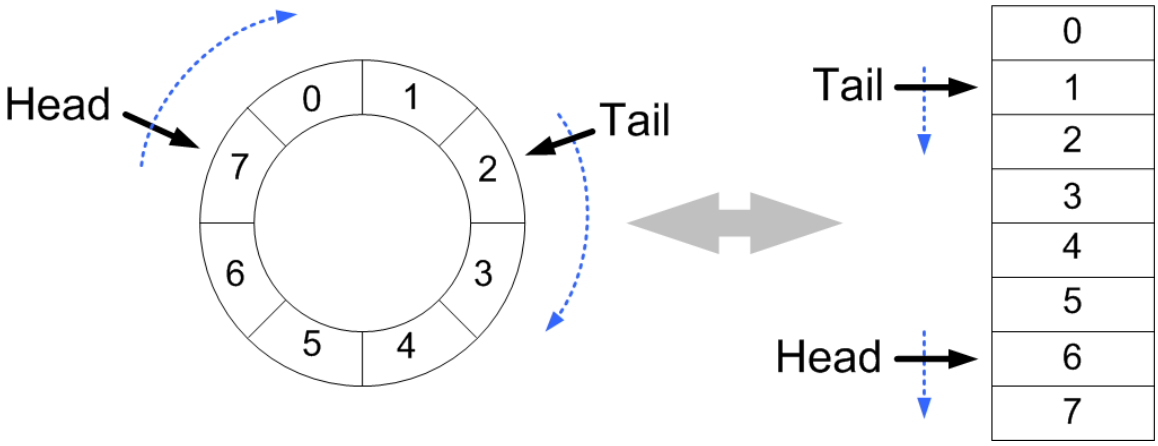


Figure 91: Circular Array

In this configuration, the head pointer is always the next location where a new packet will be inserted. The tail pointer points to the earliest ("first") packet in the buffer. Packets are read from the tail pointer. Nothing is ever inserted at the tail pointer. Similarly, packets are inserted at the head pointer, and never read from the head pointer. This gives rise to a convenient implementation in dual port RAM, where the pointers (array indices / addresses in the RAM) are the two pointers. One side always reads, and the other always writes.

When the head pointer equals the tail pointer, one of two conditions exists: either the array is completely full, and the head has wrapped around and caught the tail, or the array is completely empty, and there is no packet stored. Thus, a flag is necessary to indicate which of these two conditions exists. If the array is completely full, no additional packets can be written to the array.

Extended Data Manager. The EDM, or extended data manager is used to control special functions on the network that do not fall under the previous managers' functions. Examples of these functions are: performing a system reset, clearing the ring of traffic, forcing the node address to change, forcing the network time to change, and other things used during configuration.

Redirector. The redirector controls the dual ring fault tolerance. When an error occurs on one ring, the data is redirected to form a larger ring. Above the redirector on the LLI stack, there is no concept of dual rings, just commands coming in and going out.

Net Clock Manager. The net clock manager determines the correct value of the network clock. The correct value is based on inputs from packets coming from both sides, as well as the previous value of the network clock.

Index Manager. The index manager is responsible for keeping the received packets in phase with the transmitted packets. It predicts the delay that the node will have in transmitting and receiving, and schedules the beginning of the next transmission based on this information.

Packet Construction. There are several blocks responsible for the construction of the actual data packet. The framer takes the data in a series or array of bytes, and manages the pointer to the current byte to transmit. The CRC generator generates the cyclic redundancy check code used to verify that the packet is valid. The deframer takes a packet that has arrived, and decomposes it into pieces significant for command processing, such as the addresses, time, data and such. Depending on the packet format, the deframer will deframe the packets differently. At this level, the meaning of the command is ignored unless it is a NS_SET_PARAM packet, in which case, the packet will be processed here.

The managers are integrated into the LLI as shown in Figure 92 (single ring implementation).

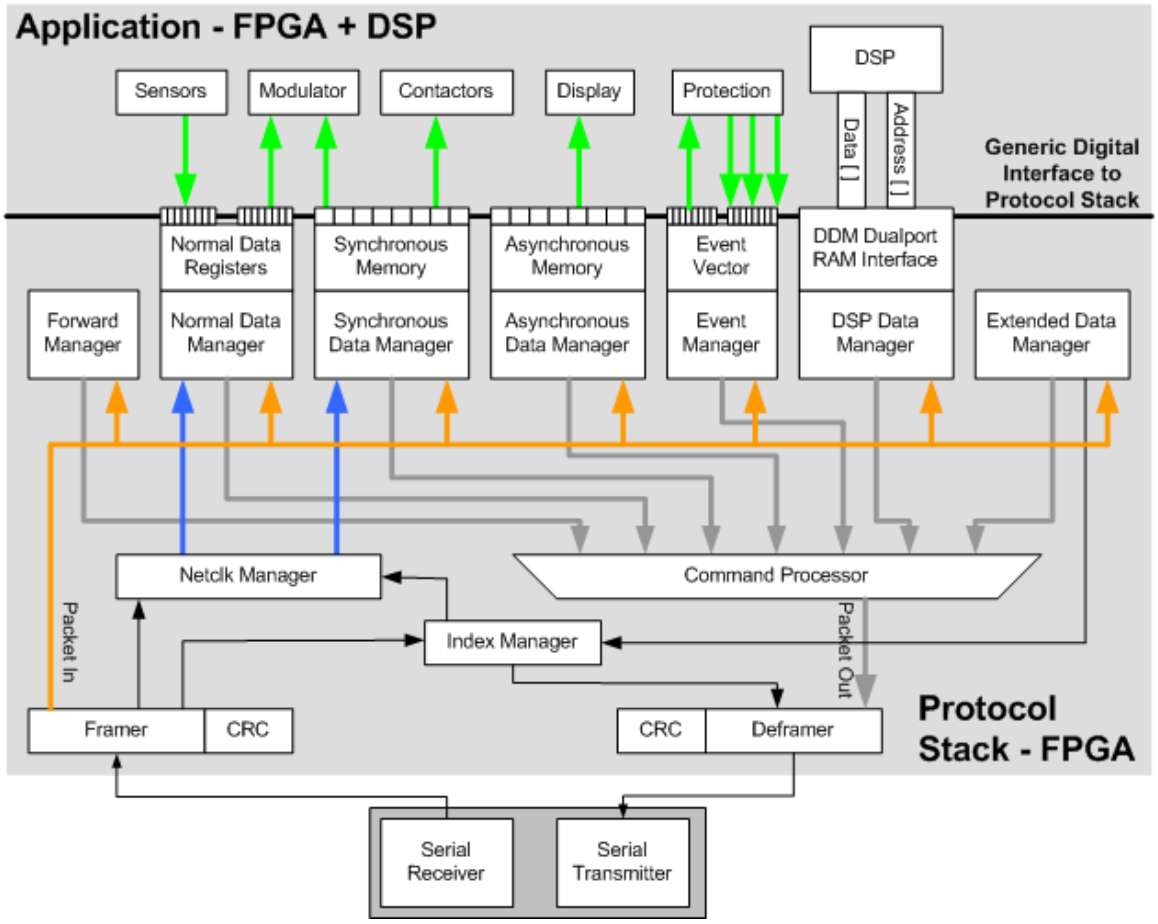


Figure 92: Single Ring PESNet LLI

In order to install the dual ring, the redirector is added. In the actual implementation, this remains to be done, but it will appear as shown in Figure 93. The redirector would just reroute packets, not process information, and therefore, it appears at the first interface at which the packets are complete.

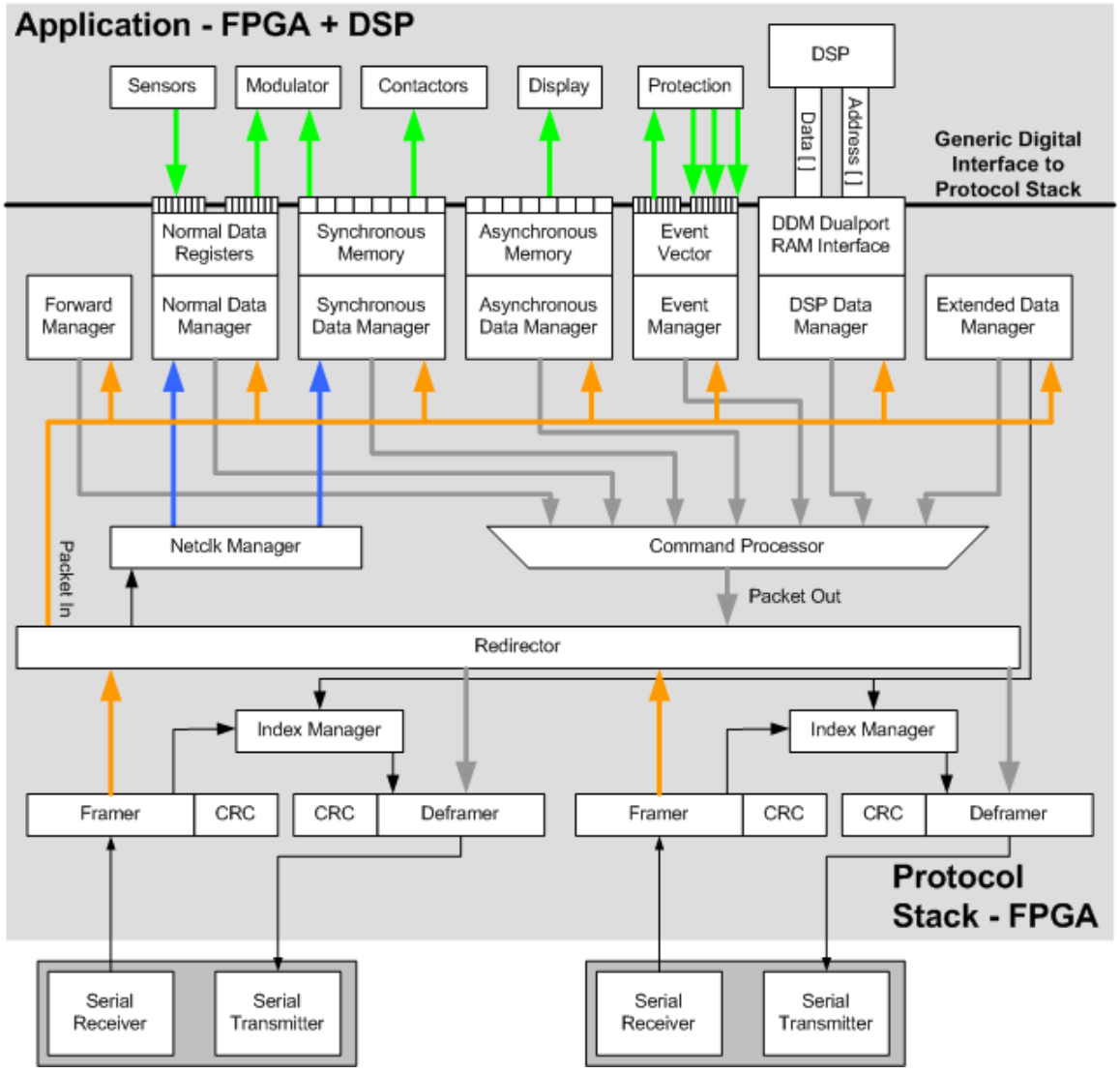


Figure 93: Protocol Stack with Redirector

6 System Test and Demonstration

The system was integrated together, and some tests were ran that showed that demonstrate the operation of one hardware manager in the test bed designed for this system. The testbed is shown in Figure 94.

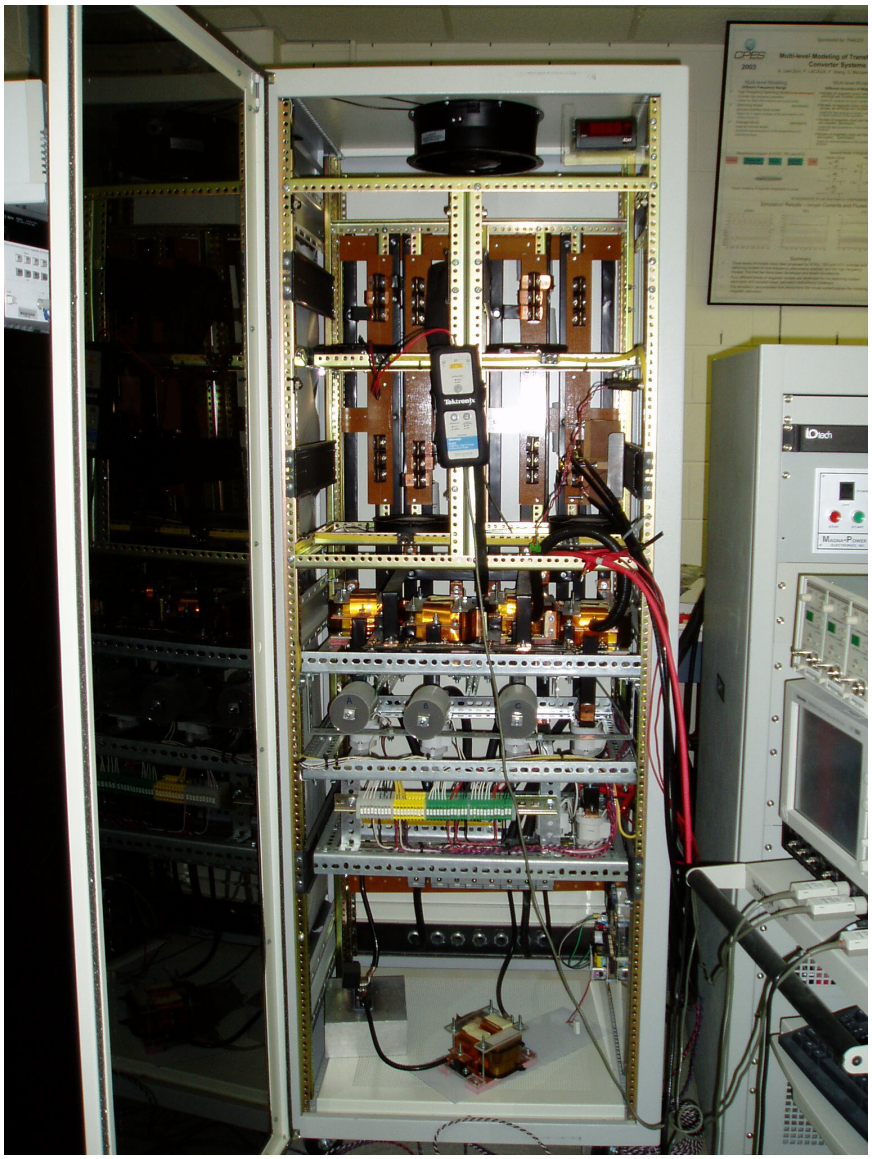


Figure 94: PEBB Plug and Play Test Bed

6.1 Hardware Manager Testing

The hardware manager tests were divided into two sections: control and functionality, and power tests.

6.1.1 DC/DC Converter Test

The DC/DC converter test was performed by creating a modulator block in VHDL so that the hardware manager only modulated only one switch, and utilized the antiparallel diode of the other switch to operate as a buck converter. The configuration of the test is shown in Figure 95.

The duty cycle was sent from a program running on the universal controller to the hardware manager over PESNet 1.2. The current was returned as a parameter that could be read in a watch window.

There were two configurations for this test. The first was to use the top switch, and connect the load resistor to ground via the inductor. The second test was to connect the load resistor to the positive DC link via the inductor, and modulate the bottom switch.

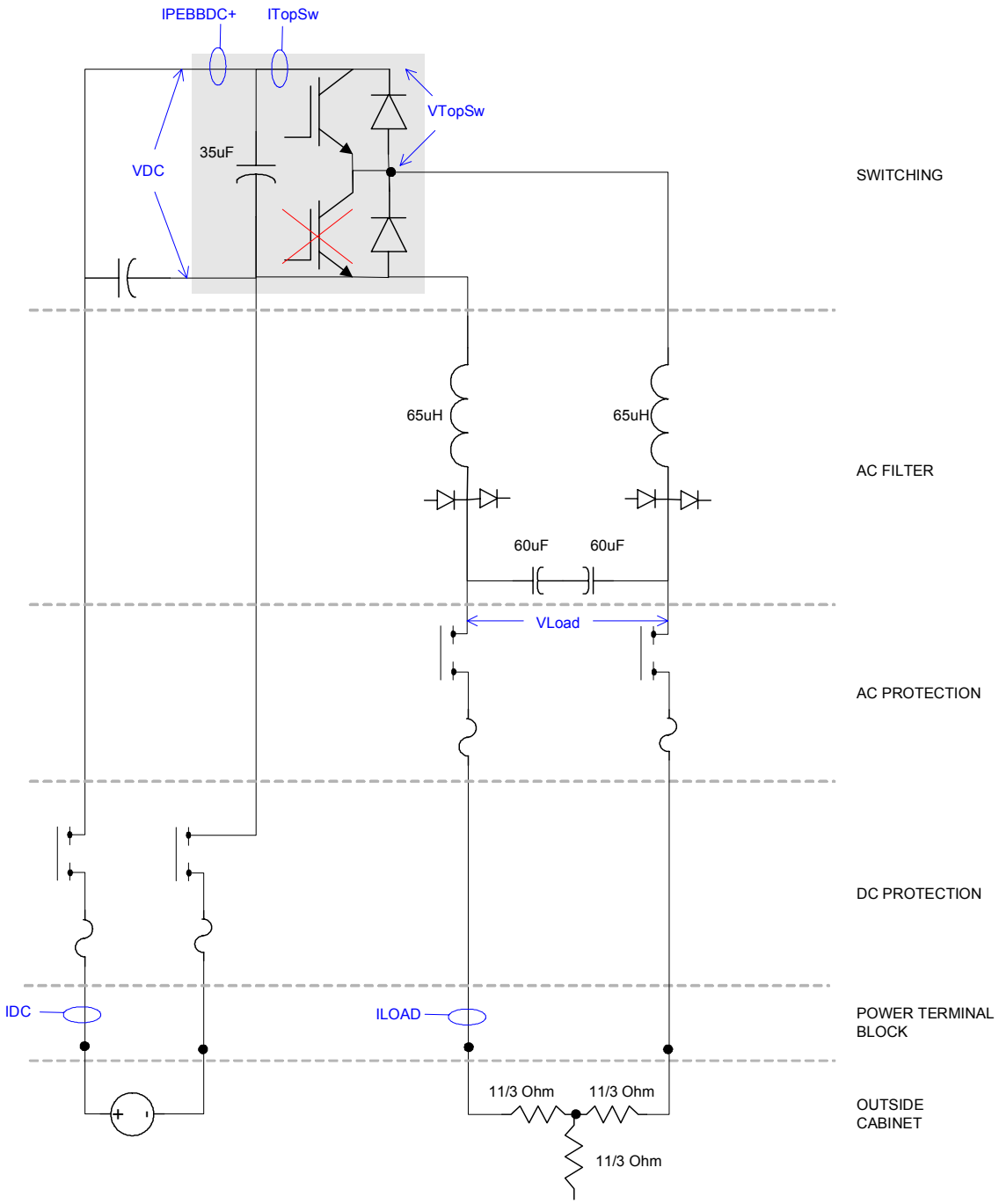


Figure 95: Test Setup

Tests were run up to a DC link voltage of 200V. The following test was the new PEBB run in a DC/DC converter mode with the bottom switch disabled at all times, and the top switch switching at 20 kHz. The configuration is the same as that shown in Figure 95. The variables measured were Vdc, VTopSw, Idc, and Vload. The only time that there is a ripple on Idc is during the top switch turn-off. The current was set at 5 A/10mV and was measured using a Tektronix Hall-Effect gun. All voltage measurements were made using Tektronix high voltage differential probes.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	5A/div	(1div = 10mV), Choke on BNC cable
4	Vload	100V/div	

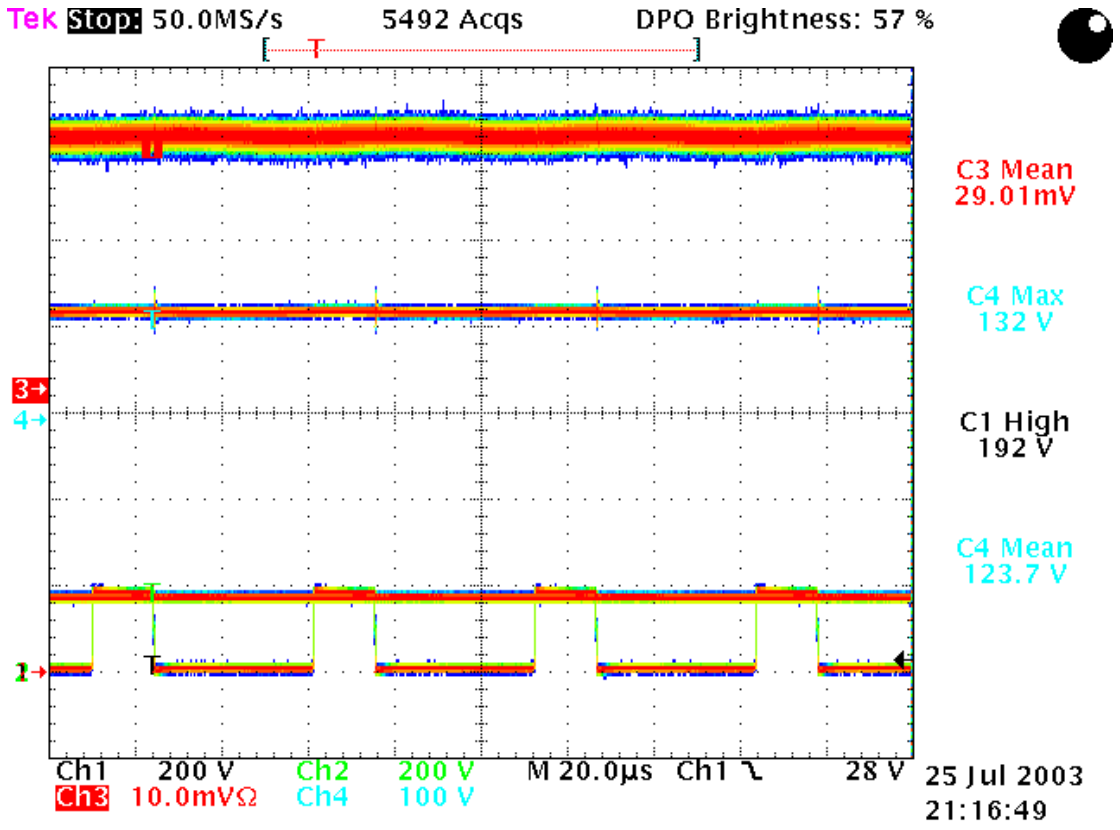


Figure 96: DC/DC Converter Switching

The turn off waveform is expanded in Figure 97. High frequency ringing can be seen here. This ringing occurs at about 15 MHz. If this high frequency ring were to travel back to the supply, it could damage it after long periods of operation. A series diode was placed to compensate for this as seen later.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	5A/div	(1div = 10mV), Choke on BNC cable
4	Vload	100V/div	

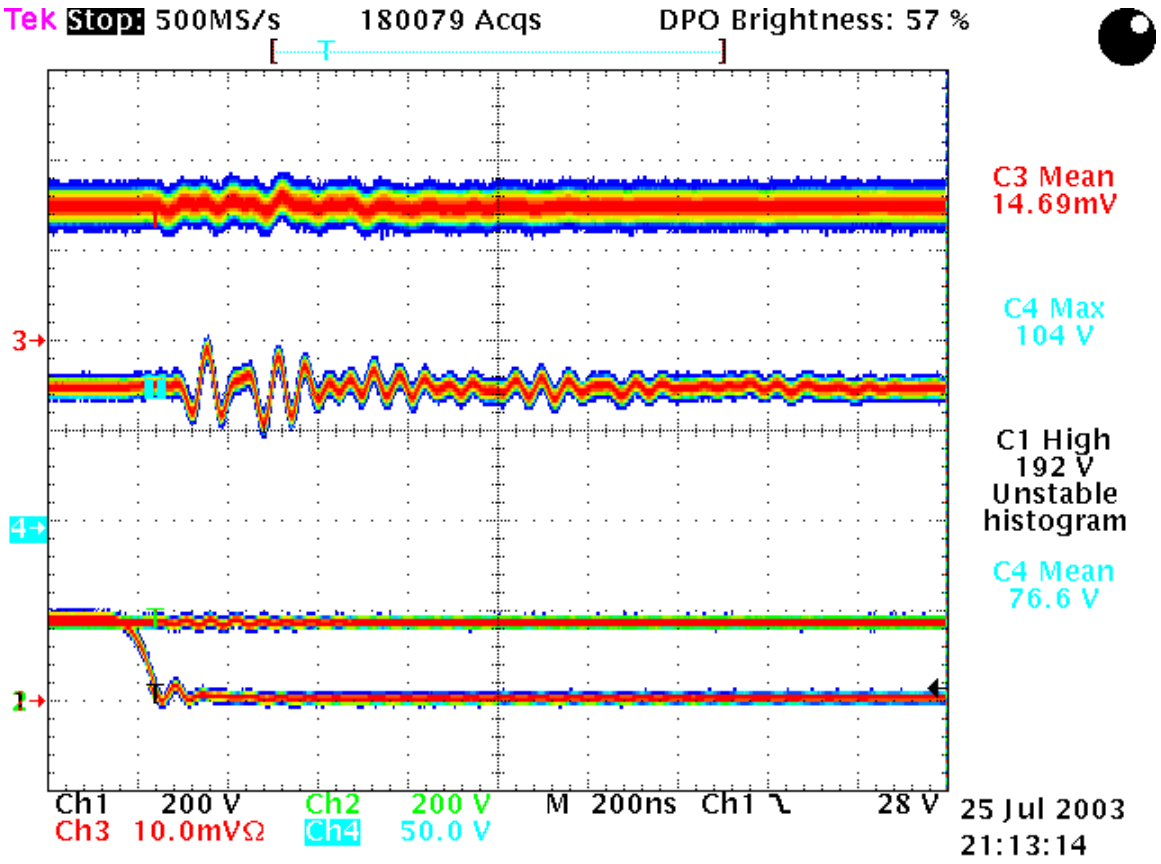


Figure 97: DC/DC Converter Switch Turn-Off

6.1.2 Pulse Test Results

A two-pulse test was performed in order to show the phase leg could safely switch at high power and EMI levels. The output of Phase C was shorted to the negative DC rail, making the load of Phase C to be the 216uH inductor. The schematic shown in Figure 98 was created using the cabinet and the new phase leg.

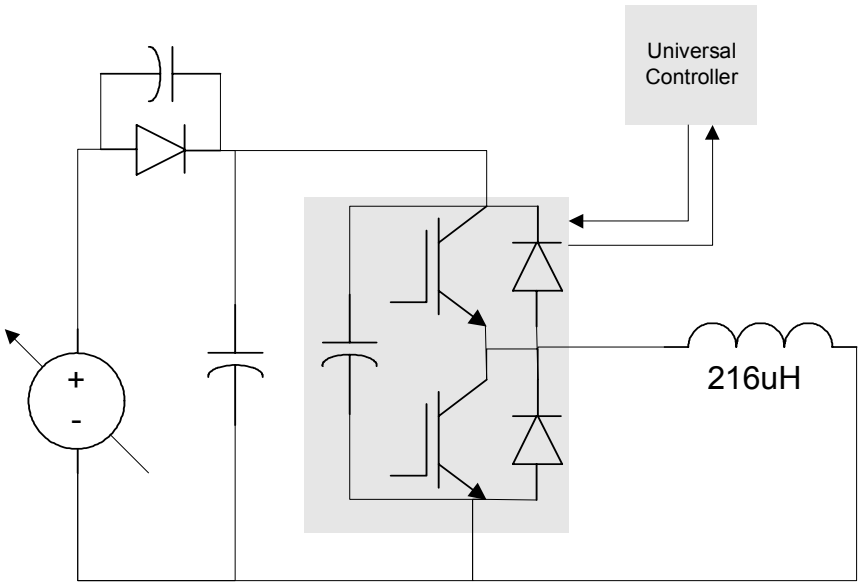


Figure 98: Pulse Test Schematic

This test consists of two pulses. The first pulse is large, with the intention of building the current up to the test level. The second pulse is a typical PWM pulse. This pulse is repeated once per second, which makes the power very small while testing large currents. The ideal switch voltage is shown in Figure 99.

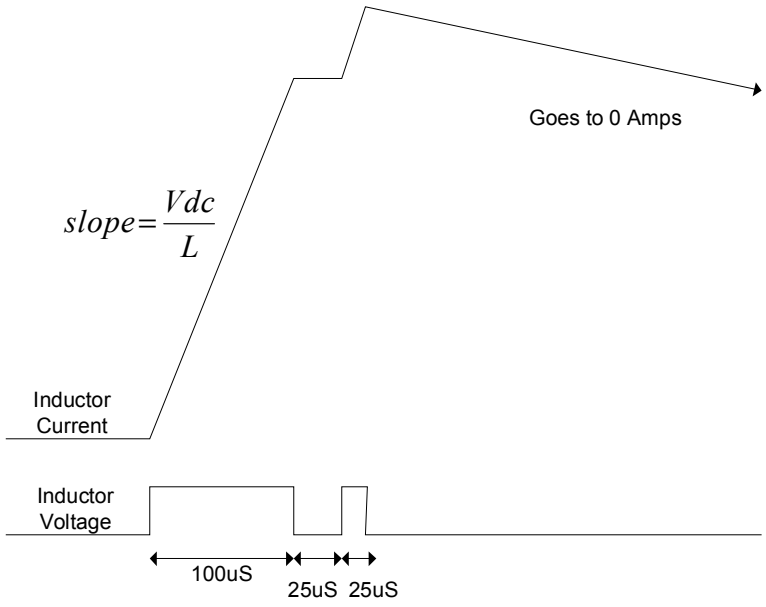


Figure 99: Pulse Test

The current peak can be calculated for turn on:

$$I = \frac{VT}{L} = \frac{150 \cdot 100\mu}{216\mu} = 69.4A$$

For turn off, the current peak is:

$$I = \frac{VT}{L} = \frac{150 \cdot 125\mu}{216\mu} = 86.8A$$

A pulse test control block was written in VHDL and used the same interface as the DC/DC converter modulator to create a waveform similar to that shown in Figure 99. There were four parameters sent to the modulator:

- The width of the first pulse
- The width of the time between the first and second pulse
- The width of the second pulse
- The period with which this pattern repeats

This was used in place of the PWM generator, and the values were controlled from the fiber optic interface via a universal controller connected to a computer using Analog Devices Visual DSP development environment. The results are shown in Figure 100.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	Zero when switch is on
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

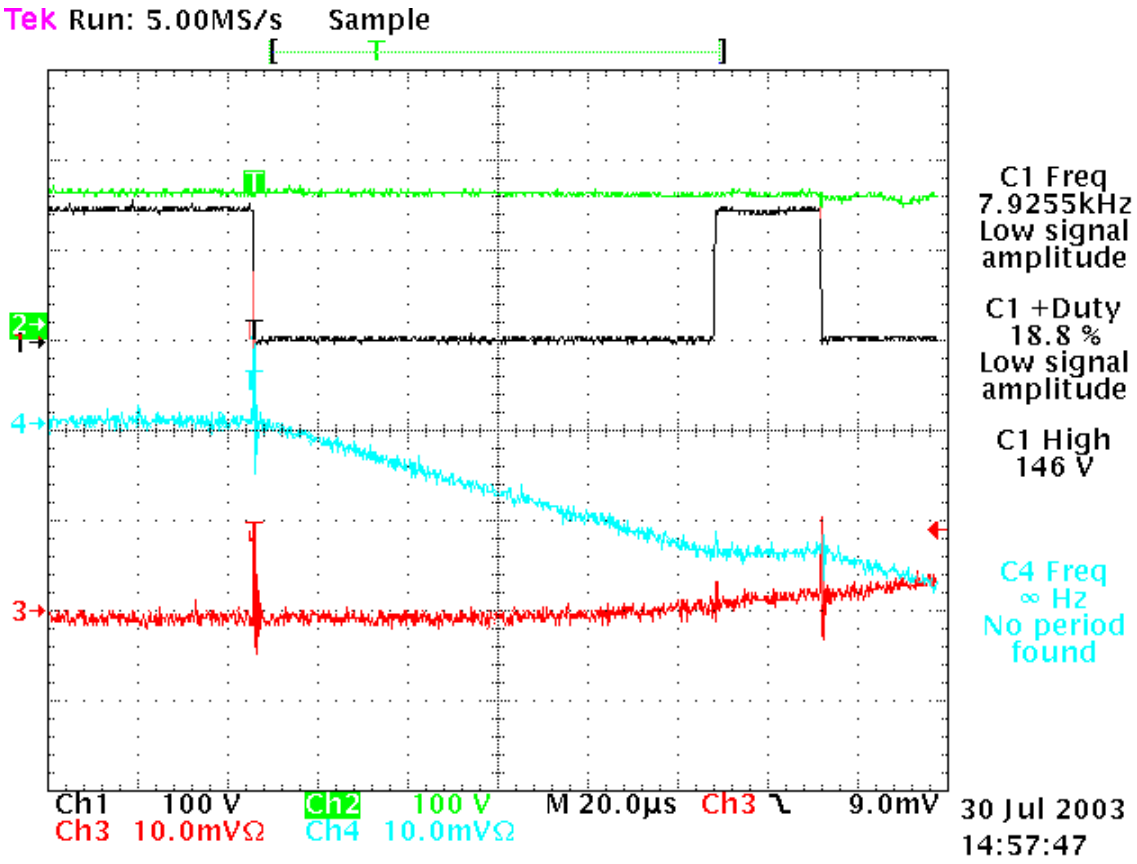


Figure 100: Pulse Test Results

When zoomed out, Figure 101 shows that the current returns to zero in 12 milliseconds.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

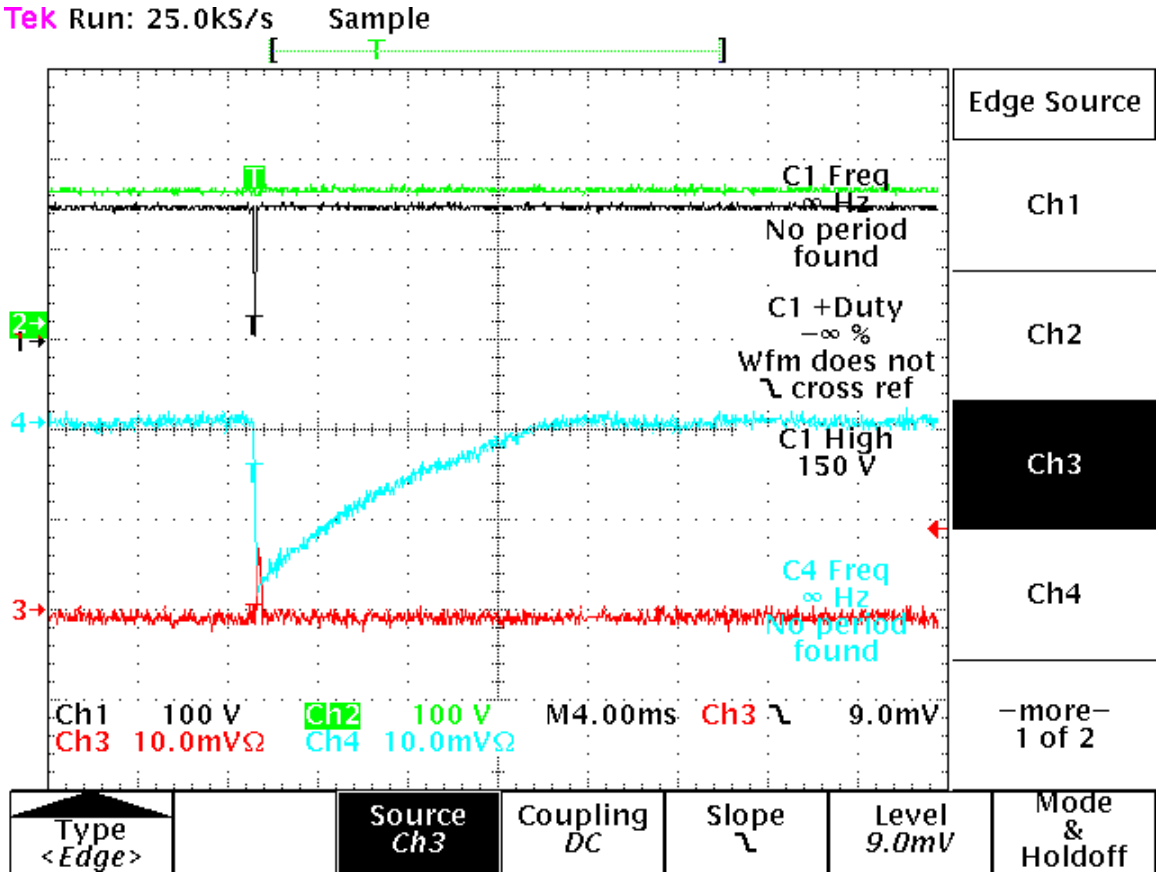


Figure 101: Pulse Test Results on Large Time Scale

A series diode was added to the system input to prevent the current from flowing back into the power supply, as some ringing was encountered. The diode can be seen blocking the negative current in Figure 102.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

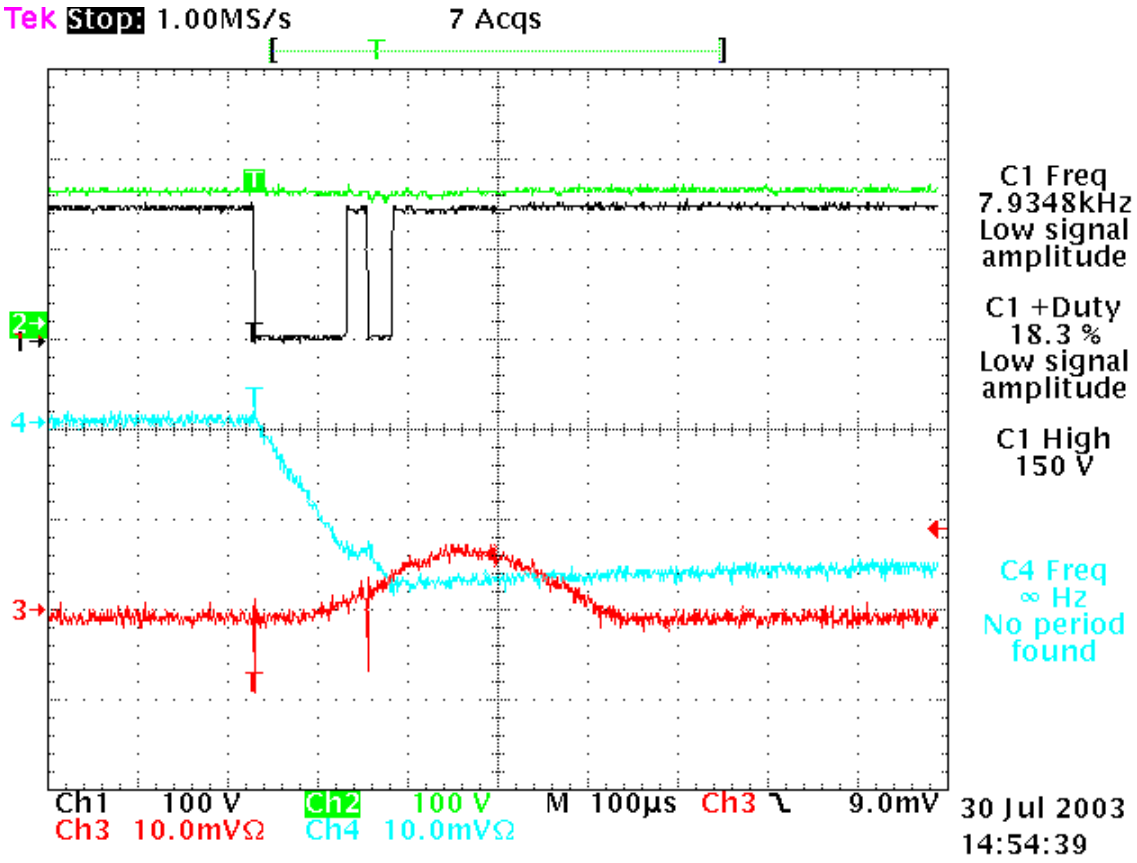


Figure 102: Effect of Series Diode

The diode can be seen blocking the reverse current as shown in Figure 103. It is also noted that the pulse width was changed, and the current was tested at a peak value of 150 Amps with consistent results.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDiode	10V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

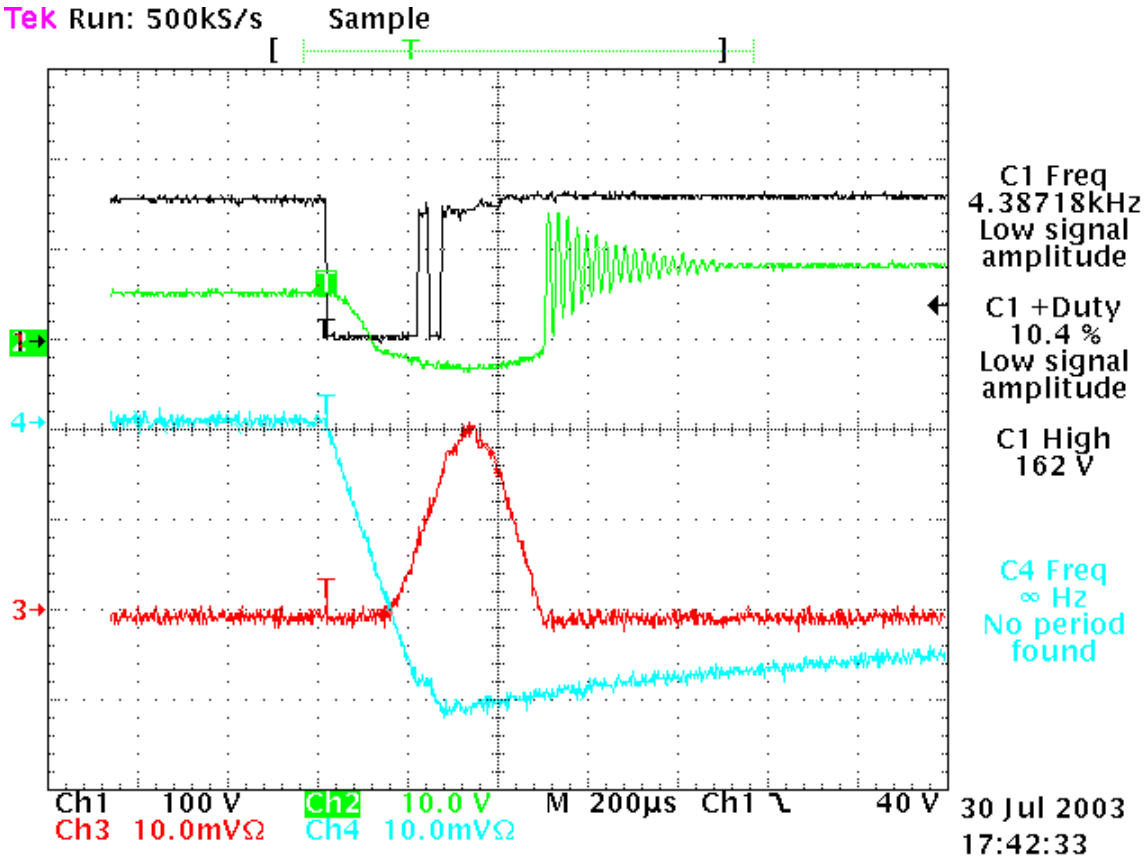


Figure 103: Diode Blocking Reverse Current

7 Conclusions and Future Work

7.1 Conclusions

A control architecture has been developed as a response to the requirements of advanced power electronics systems for high power converters. Existing control architectures were considered, focusing on examples from manufacturing automation, and the restrictions on each were identified.

Details about the control architecture were described, including a universal controller and a hardware manager. The controller was a power hardware independent supervisor with onboard features designed as a result of defining usability, development, and application requirements. The controller is responsible for implementing control algorithms that require information from more than one controller. The controller was specified, designed, manufactured, and tested. VHDL code was developed for the onboard FPGA to support the controller hardware functionality. A second version was created based on issues identified in the first version. The second version was also designed, manufactured and tested. VHDL code was updated to support the new functions. Project management and quality techniques were used throughout the device lifecycle to keep track of defects and possible new features in future improvements. Systematic team-based approaches were used to provide quality checkpoints and design reviews.

The hardware manager is specific to the hardware being controlled, and is supervised by one or more universal controllers. The hardware manager is responsible for the modulation scheme and local control functions that are specific to the hardware being controlled. Application examples were analyzed, and a list of requirements was generated. Based on these requirements, a design specification was created describing the hardware manager. The hardware manager was implemented and manufactured, and system tests were shown that verify the operation of the entire system under PESNet 1.2.

The existing version of PESNet was described and analyzed. Based on the results of this analysis, a detailed specification for a future version of PESNet (2.2) in development was given for commands and modular implementation from the physical layer to the application layer interface for both hardware managers and universal controllers. Requirements and extensions necessary to support complete plug and play operation were given.

7.2 Future Work

7.2.1 Controller

DAC Output Filter. A late realization was that the DAC has a large overshoot / undershoot when the value is changed, even if by a small amount. The settling time is very high, but this overshoot appears on scope waveforms. An output filter can be used to greatly reduce this effect.

Reconfiguration Over PESNet. It is desirable to reprogram parts of the FPGA over PESNet, such as application specific parts, or for the purpose of upgrading.

Sending DSP Program Over PESNet. It is possible to load the DSP from PESNet, which is useful in a multiple DSP system, where one controller is connected to the debugging station, and parameterizes the system over the PESNet network. This will reduce the development cycle time, as well as reduce the number of JTAG emulators or flash writes necessary.

DAC Connector. The digital to analog converter pins are accessed via a straight connector. When multiple boards are stacked together, these pins become less accessible. Using a right angle connector and changing the orientation of the connector can improve accessibility to these pins.

Peripheral Flash Location. Currently, the location of the peripheral flash prevents it from being socketed while the controller is using the PMC bus. By moving the peripheral flash to a different location, this problem can be avoided.

Communication LEDs. The use of LED indicators on the controller to indicate PESNet status would be helpful for debugging and visual indication of status. It is of relatively small effort to integrate these.

7.2.2 Hardware Manager

Dynamic Reconfiguration. As seen in chapter 6, the code in the hardware manager FPGA was changed manually to perform different types of tests. However, if there was a way to dynamically change the behavior of the modulator, the hardware manager code would be less application specific, allowing the application to download these parts of the hardware manager firmware on startup. One approach here may be to make containers that would hold code that could be executed, such as a small instruction processor, such as the Microblaze processor by Xilinx, which is a processor written in VHDL that can be implemented in the FPGA, and executes custom instructions. A second approach would be to allow the FPGA to undergo partial reconfiguration.

Current Source Adaptation. The primary focus of this work has used voltage source topologies for research. However, another class of converters exists, referred to as current source converters. These converters use series diodes with the switches. It is possible to develop a module that can be plugged on to a voltage source PEBB to make it a current source, as show in Figure 104. This module would go in series with the power semiconductor and the rest of the converter system.

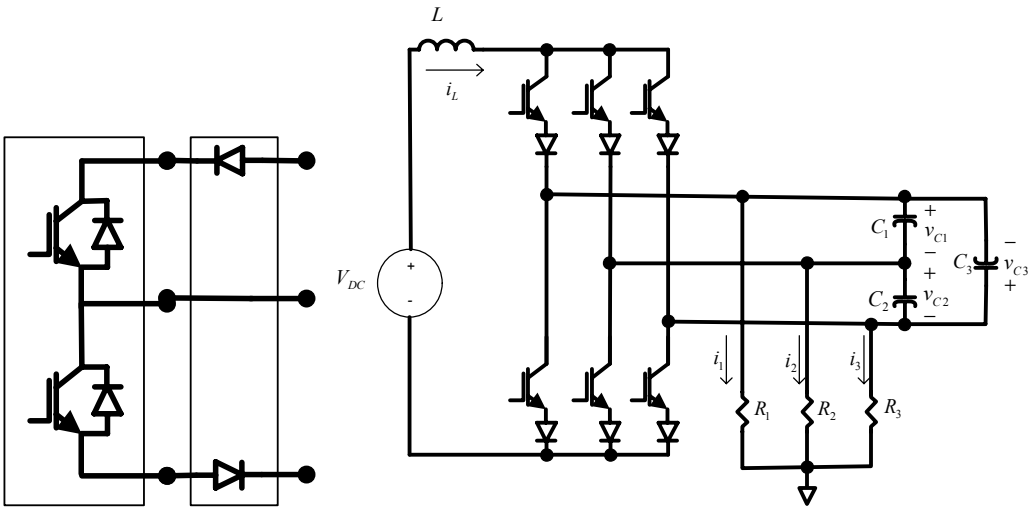


Figure 104: CSC Diode Adapter (left) and CSC (right)

A second improvement may not require hardware development. It may be possible to dynamically reconfigure the hardware manager to allow for custom code execution on the HM where the code is downloaded from another configuration node, such as a universal

controller via PESNet. This would allow for configurable protection systems, as well as configurable modulation strategies, such as the use of sigma-delta modulation or phase-shift modulation.

Hardware Manager Stacking. Occasionally, there is an application with advanced requirements at the hardware manager and PEBB level. The requirements may be more than what a single hardware manager may be able to handle. A standardized interface may be desired at the hardware manager level to add support for a plug-in card containing a small DSP, FPGA, or another hardware manager. There is currently an expansion connector on the hardware manager, but a stacking connector may also be beneficial.

7.2.3 PESNet

Use of PESNet as higher level communications protocol. PESNet has been demonstrated at the converter level. Converters exist as parts of larger systems. This is referred to as the station level in [23]. This feature is desirable in several systems, including ship systems with advanced control architectures, like those proposed in [76] and in [6].

It is possible to allow converters on different rings to be controlled from a supervisory network. PESNet may be able to be used at this level as well. Due to the nature of the time constants involved here, it may be desirable to change some details to make the protocol more suitable for higher levels. The temporal distribution of the system allows for station level communication networks to have less stringent deadline requirements than at the converter level.

Expanding From A Single Ring Architecture. Currently, PESNet exists in a ring architecture. If it is used in a higher level, a new topology may be desired. As more nodes appear on the ring, there exists a tradeoff between node count, data size, and switching cycle at the same frequency. Eventually, a mesh network would work well.

As a step toward a mesh network, a hierarchical ring structure is proposed. The structure of this network is shown in Figure 105.

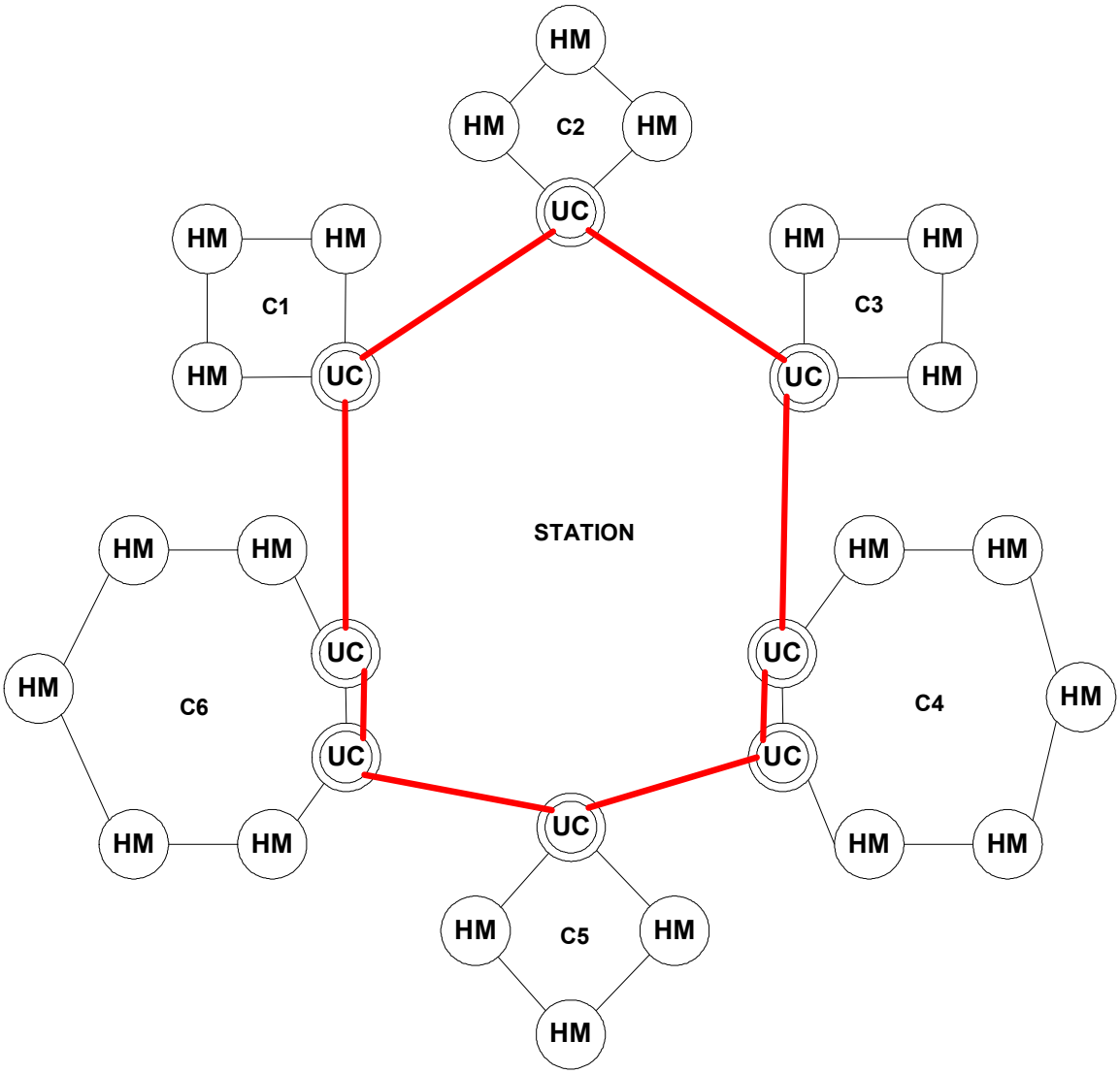


Figure 105: Hierarchical Ring Network

The current system can demonstrate a hierarchical network, however, there needs to be controllers stacked together in the central ring to add enough PESNet ports.

PESNet is not currently a routable protocol. This means that communication on the same network is possible between nodes on the same ring. It may be possible to add a routing field to the packet to allow data to switch rings. However, this makes the protocol

complex, as it adds complexity at the lower levels of the protocol such as routing tables and such.

The use of a hierarchical ring architecture may allow for multiple failures on the ring, which should be a desirable property of a system level control network, as the network is more spatially distributed than a single converter network.

Moving to 8B/10B Encoding. Currently, 4B/5B encoding is used. This code is less reliable than using a 8B/10B code due to the imbalance within the optical to electrical receiver [77]. The use of 8B/10B encoding still keeps the data characters separated from the command characters. Cypress makes chips that are very similar to the TAXI chips, with the exception that these support the 8B/10B protocol.

Decentralized Initializaton. The current proposed version of PESNet requires a bus master to initialize the communication bus and set bus parameters, such as the packet type and network clock. It may be possible to negotiate this information in a decentralized way, removing the dependence on a central bus master.

Plug and Play Extensions. Overlays need to be created for device types that allow plug and play operation of the power stage via the control software and hardware modules. One approach would be to describe module types with text files containing module parameters such as what was done in Profibus using GSD files. These files would give a description of the memory overlay, and what information was stored at what offset. Generic module drivers could utilize this information to control these modules regardless of the implementation details.

Integrated Transceiver Functions into FPGA. As FPGAs become more advanced, they are getting features such as processors embedded into the silicon, as well as transceivers. The new Virtex-II Pro FPGAs from Xilinx have multiple transceivers and processors built into the FPGA chip. It would be good to eliminate dependency on chips by implementing these lower level protocol details in the FPGA.

High Resolution Timing. For topologies with very fast time constants, it is important to ensure that nodes are synchronized as close as possible. Examples of such configurations would be multilevel converters such as NPC which are built using combinations of hardware managers, as shown in Figure 106.

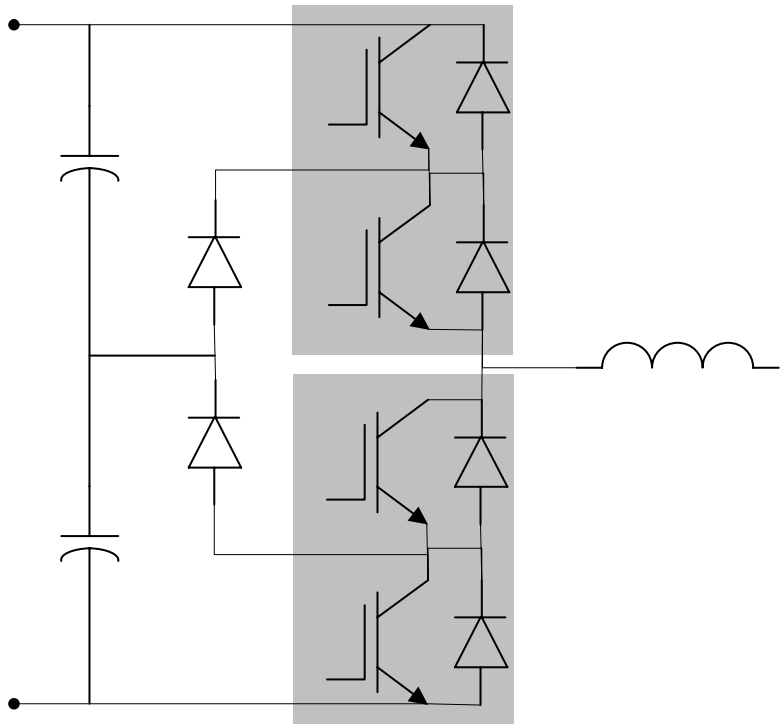


Figure 106: 3-Level NPC Phase Leg using 2 Hardware Managers

An even more challenging configuration would be parallel switch topologies, or independent soft-switching topologies, where two hardware managers are used to construct a single phase leg. An example of a ZCT soft-switching converter [78] is shown in Figure 107. The synchronization between these phases may require resolution that is higher than the network clock. Even though the synchronization jitter is small, the granular nature at the packet level prevents fine operation of the converter. It may be possible to take advantage of internal timers to address this.

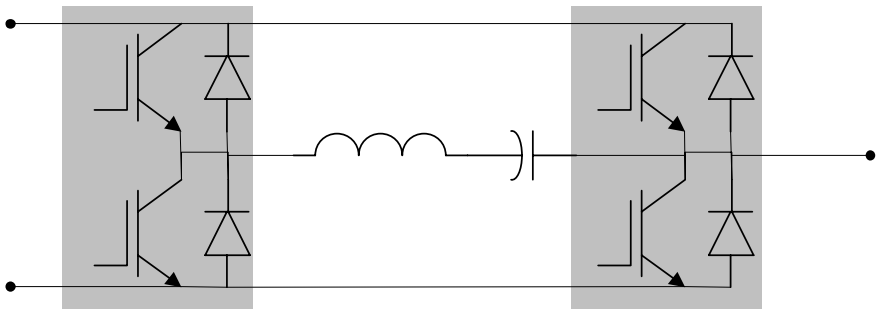


Figure 107: ZCT With 2 Hardware Managers

Another issue is current source converters built using per phase hardware managers. The current source converter synchronization must be tightly coupled to avoid turning off all switches on one side, which would be a sudden interruption of energy flow.

One goal of the converter system is to enable plug and play operation of power conversion systems. In order to achieve this, there should be some classification of devices on the PESNet network, similar to what is done in PCI and CardBus type systems [79], [80]. In order to enable this, device classes need to be created that allow for common configuration of devices without vendor dependence.

8 References

[1]Burgos, R.P.; Wiechmann, E.P.; Rodriguez, J.R. "An adaptive fuzzy logic controller for three-phase PWM boost rectifiers: design and evaluation under transient conditions", Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE, Vol.2, Iss., 31 Aug-4 Sep 1998 Pages:761-767 vol.2

[2] Xepapas, S., Kaletsanos, A., Xepapas, F., Manias, S., "Sliding-mode observer for speed-sensorless induction motor drives", IEE proceedings on Control Theory and Applications, Volume 150, Issue 6, Nov. 2003

[3] Chester Petry, "The Electric Ship and Electric Weapons", Presentation. NDIA 5th Annual System Engineering Conference, Tampa, FL, October 22-24, 2002

[4] Narain Hingorani and Laszlo Gyugyi Understanding FACTS: Concepts and Technology of Flexicble AC Transmission Systems. Institute of Electrical and Electronics Engineers, Inc. New York, New York 2000

[5] ABB, "Dynamic Voltage Restorer for a Critical Manufacturing Facility," Presentation at IEEE/PES Transmission and Distribution Conference, September 2003

[6] Ed. Zivi, "ONR Control Challenge White Paper", US Naval Academy (available from the Electric Power Network Efficiency and Security website, http://www.usna.edu/EPNES)

[7] CDR. John Amy Jr., "Considerations in the Design of Naval Electric Power Systems", IEEE Power Engineering Society, Summer Meeting, 2002, Volume I, 21-25 July, 2002, pages 331-335.

[8] Hirofumi Akagi, "High Power Applications of Power Electronics In Japan", CPES Annual Seminar, 2003, Blacksburg, Virginia 2003

[9] Hirofumi Akagi, "Large Static Converters for Industry and Utility Applications", Proceedings of the IEEE, Volume 89, Issue 6, June 2001

[10] Technology for Future Naval Forces: Chapter 8: Electric Power and Propulsion, http://www.nap.edu/html/tech_21st/t8.htm

[11]Xilinx Inc., XAPP151: Virtex Series Configuration Architecture Users Guide, available at http://www.xilinx.com/bvdocs/appnotes/xapp151.pdf

[12] Terry Ericsen, Albert Tucker, David Hamilton, George Campisi, Clifford Whitcomb, Joseph Borraccini, William Jacobsen, "Standardized Power Switch System Modules (Power Electronics Building Blocks)", Available from AEPS Website (http://aeps.onr.navy.mil)

[13] Zhang, Richard. High Performance Power Converter Systems for Nonlinear and Unbalanced Load/Source. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA 1998.

[14] Oyama, J., Xiarong Xia, Higuchi, T., Yamada, E. "Displacement angle control of Matrix Converter" Power Electronics Specialists Conference, 1997. PESC '97 Record., 28th Annual IEEE, Vol.2, Iss., 22-27 Jun 1997, Pages:1033-1039 vol.2

[15] Jia Wu, Implementation of a 100kW Soft-Switched DC Bus Regulator Based on Power Electronics Building Block Concept, Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 2003

[16] Kun Xing, Modeling, Analysis, and Design of Distributed Power Electronics System Based on Building Block Concept, Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999

[17] Lee, Dong-Ho. A Power Conditioning System for Superconductive Magnetic Energy Storage based on Multi-Level Voltage Source Converter. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999

[18] Matt Superczynski, Analysis of the Power Conditioning System for a Superconducting Magnetic Energy Storage Unit, Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2000

[19] Celanovic, Nikola. Space Vector Modulation and Control of Multilevel Converters. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA 2000

[20] Josep Pou and Dushan Boroyevich, "A New Feed-Forward Space-Vector PWM Method to Obtain balanced AC output voltages in a three-level neutral-point clamped converter", IEEE transactions on Industrial Electronics, Volume 49, Issue 5, October 2002, pages 53-58

[21]Celanovic, I.; Milosavljevic, I.; Boroyevich, D.; Cooley, R.; Guo, J., "A new distributed digital controller for the next generation of power electronics building blocks", IEEE Applied Power Electronics Conference and Exposition, 2000, Volume 2, pages 889-894, February 2000

[22]Rosado, S. Wang, F. Boroyevich, D. Wachal, R., "Control interface characterization of power electronics building blocks (PEBB) in utility power system applications", Power Engineering Society General Meeting, 2003, IEEE , Volume: 3 , 13-17 July 2003

[23] Dushan Boroyevich et al., "Standard Cell Open Architecture Power Conversion Systems: Proposal to the Office of Naval Research", Blacksburg, VA 2001

[24] Tom Stanley, PCI System Architecture, Addison Wesley, Boston MA, 1999

[25] Art Baker, The Windows NT Device Driver Book, Prentice Hall PTR, Upper Saddle River, New Jersey 1997

[26] David Corman and Jeanna Gossett, "Weapons System Open Architecture – A Bridge Between "Embedded" and "Off-Board"", IEEE AESS Systems Magazine, November 2002

[27] Wonderware InControl website (http://www.wonderware.com)

[28] Allen-Bradley ControlLogix Website (http://www.ab.com)

[29] GE Fanuc CIMPLICITY website (http://www.geindustrial.com/cwc/gefanuc/ctrlio.html)

[30]Deutsches Institut fur Normung, DIN 19245 Standard,

[31] Profibus Workshop Training Guide, Profibus Interface Center, Johnson City, Tennessee, 1997

[32] Profibus International website (http://www.profibus.com)

[33] Open Devicenet Vendor's Association website (http://www.odva.org)

[34] AS-International Association (http://www.as-interface.com)

[35] SERCOS North America (http://www.sercos.com)

[36] Beckhoff Lightbus (http://www.beckhoff.com/english/default.htm?lightbus/default.htm)

[37] Internaltional Organization for Standardization, ISO 7498 standard

[38] Jeffrey Burl, Linear Optimal Control: H2 and H[Infinity] methods, Addison Wesley Longman, Menlo Park, California 1999

[39] Hiti, Silva Three Phase PWM Converter Modeling and Control, Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA 1995

[40] Ericson, Robert and Maksimović, Dragan, Fundementals of Power Electronics: Second Edition, Kluwer Academic Publishes, Norwell, Massachusetts 2001

[41] H. Prasad, Analysis and Comparison of Space Vector Modulation Schemes for Three-Leg and Four-Leg Voltage Source Inverters, Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 1997

[42] Jose Rodriguez, Jih-Sheng Lai, Fang Zheng Peng, "Multilevel Inverters: A Survey of Topologies, Controls, and Applications" IEE Transactions on Industrial Electronics, Vol. 49, No. 4, August 2002

[43] Madhav Devidas Manjrekar, Topologies, Analysis, Controls, and Generalization in H-Bridge Multilevel Power Conversion. Dissertation, University of Wisconsin – Madison, Madison, Wisconsin 2000.

[44] Dushan Boroyevich, "Power Electronics Building Blocks Hardware and Software Control Architectures" ONR Advanced Electric Power Systems Review and Workshop, Ocean City, MD, December 2001

[45] Nutt, Gary. Operating Systems: A Modern Perspective, Addison-Wesley Longman, Reading, Massachusetts 2000

[46] Kai Hwang and Zhiwei Xu, Scalable Parallel Computing, McGraw Hill, Boston, Massachusetts 1998

[47] James Armstrong and Gail Gray, Structured Logic Design With VHDL. Prentice Hall PTR, Upper Saddle River, New Jersey, 1993

[48] Kevin Skahill, VHDL for Programmable Logic, Addison-Wesley, Reading, Massachusetts 1996

[49] M. Morris Mano, Computer System Architecture, Prentice Hall, Englewood Cliffs, New Jersey 1993

[50] Nelson, Nagle, Carroll, Irwin, Digital Logic Circuit Analysis and Design, Prentice Hall, Englewood Cliffs, New Jersey 1995

[51] Xilinx, Inc. "Data Sheet DS083-2: Virtex-II Pro Platform FPGAs: Functional Description", available from Xilinx (http://www.xilinx.com)

[52] Xilinx, Inc. "Application Note XAPP290: Two Flows for Partial Reconfiguration: Model Based or Difference Based", November 2003

[53] Parool Mody, Supporting Transparent Distributed Messaging for Dataflow Applications in Power Electronics Control Systems, Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2003

[54] Xilinx, Inc. "Data Sheet DS026: XC18V00 Series In-System Programmable Configuration PROMs", December 15, 2003

[55] Xilinx, Inc. "Application Note XAPP138: Virtex FPGA Series Configuration and Readback", July 2002

[56] AD8582 +5 Volt, Parallel Input Complete Dual 12-Bit DAC Data Sheet, Available from Analog Devices, Inc. (http://www.analog.com)

[57] TIL311 Hexadecimal Display With Logic Data Sheet, Available from Texas Instruments, Inc. (http://www.ti.com)

[58] IEEE Specification for Common Mezzanine Card (CMC), p1386

[59] Interbus-S module datasheet for IBS CT 24 IO GT-T Coupling module (Gateway), available from Phoenix Contact (http://www.phoenixcontact.com)

[60] Microsoft CEPC specification, available from Microsoft Developer Network (http://www.msdn.microsoft.com)

[61]Am29LV040 Datasheet, Available from Advanced Micro Devices (http://www.amd.com)

[62] ADSP-21160 Datasheet, Available from Analog Devices Inc. (http://www.analog.com)

[63] Analog Devices ADSP-21160 Hardware Reference, available from Analog Devices, Inc. (http://www.analog.com)

[64] EZ-Kit Reference for ADSP-21160 DSP available from Analog Devices, Inc. (http://www.analog.com)

[65] Milosavljevic, Ivana. Power Electronics System Communications. Thesis, Virginia Polytechnic Institute and State University. Blacksburg, Virginia

[66] Ivan Celanovic, A Distributed Digital Control Architecture for Power Electronics Systems, Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia

[67] HP/Agilent Application Note 1066, available from http://www.agilent.com

[68] Steve McConnell, Software Project Survival Guide, Microsoft Press, Redmond, Washington 1998

[69] Lewis, Project Manager's Desk Reference, 2nd Edition McGraw Hill, Boston, Massachusetts, 2000

[70] Schertz and Whitney Design Tools for Engineering Teams, Delmar, Albany, New York 2001

[71] Evans and Lindsay, Management and Control of Quality, 4th Edition. South-Western College Publishing, Cincinnati, Ohio 1999

[72]Bin Zhang, Huang, A.Q., Bin Chen, Yunfeng Liu, "A new generation emitter turn-off (ETO) thyristor to reduce harmonies in the high power PWM voltage source converters", Power Electronics and Motion control conference, 2003, IPEMC 2004, 48th international, Volume 1, pages 327-331, Aug 14-16th 2004.

[73] Sirisukprasert, S.; Huang, A.Q.; Lai, J.-S., "Modeling, analysis and control of cascaded-multilevel converter-based STATCOM", Power Engineering Society General Meeting, 2003, IEEE , Volume 4, 13-17 July 2003

[74] Shepard, Steven, SONET/SDH Demystified, McGraw Hill, New York, New York 2001

[75] Cypress Semiconductor, "CY7C9689-AC Data sheet", Available from Cypress Semiconductor (http://www.cypress.com)

[76] Ericsen, Terry Advanced Electrical Systems

[77] Benner, Alan. Fibre Channel for SANs, McGraw Hill, New York, New York 2001

[78] Luca Solero, Dushan Boroyevich, Yong Li, and Fred Lee "Design of Resonant Circuit for Zero-Current-Transition Techniques in 100-kW PEBB Applications", IEEE Transactions on Industrial Applications, Vol 39 No. 6, November 2003

[79] Tom Shanley and Don Anderson, PCI System Architecture, 4th Edition, Addison-Wesley, Inc. Boston, Massachusetts 1999

[80]Tom Shanley, Plug and Play System Architecture, McGraw Hill, New York, New York 1995

9 Appendices

9.1 PCB Manufacturing Specifications

When manufacturing the controller, it is important to specify attributes about the module. The following are some of the attributes specified when ordering the controller.

Table 15: Universal Controller PCB Manufacturing Specifications

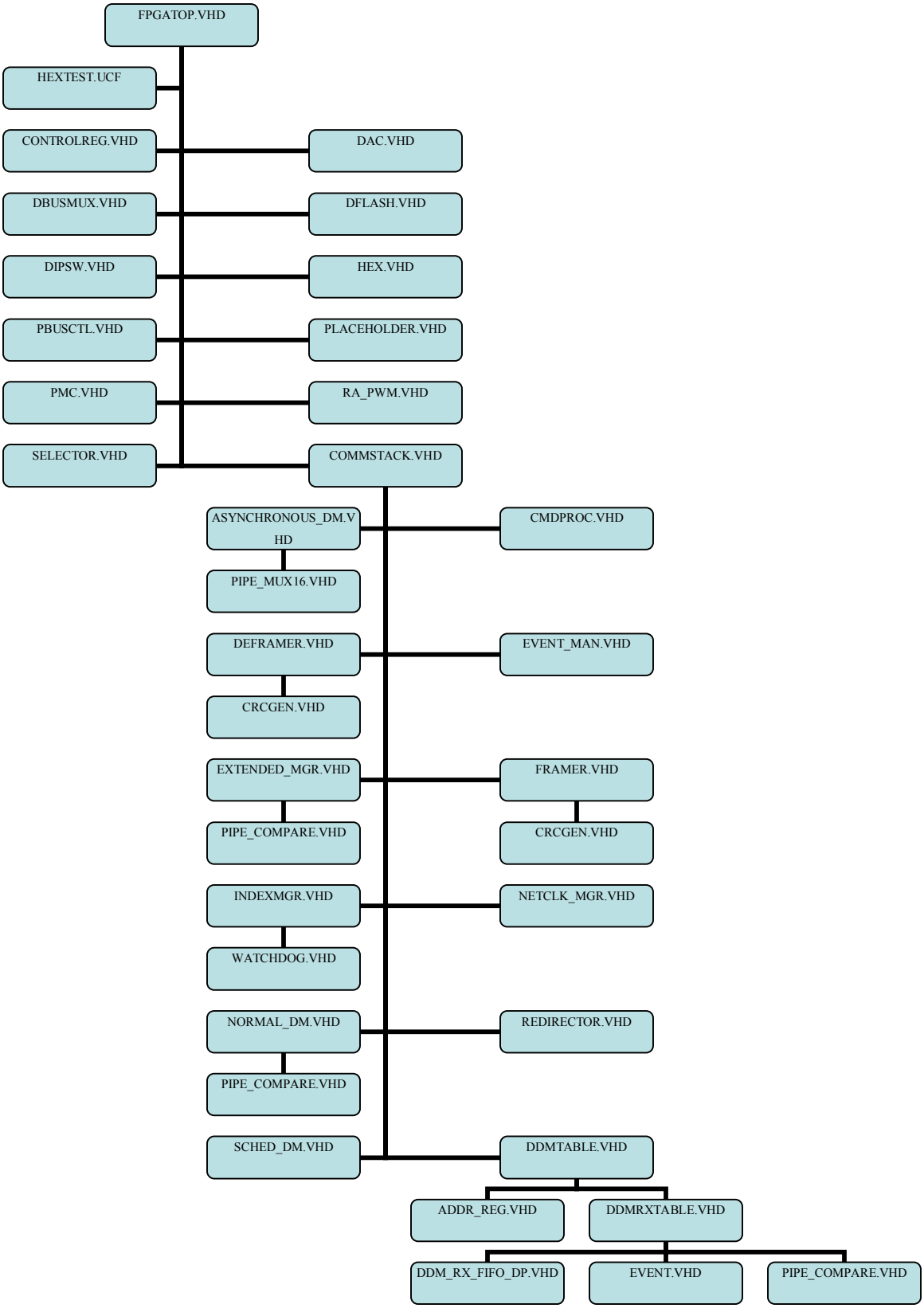
Parameter	Description	Value
PCB Thickness	The thickness of the whole PCB. This is constrained by the PMC spec (IEEE p.1386)	0.062”
Minimum Trace Width	The width of the smallest trace on the PCB	0.005”
Minimum Space	The minimum clearance between different nets on the PCB	0.005”
Copper thickness (outside)	The thickness of the copper on the outer layers	1 oz.
Copper thickness (inner)	The thickness of the copper on the inner layers	0.5 oz.
Finish		HASL
Mask	The type of silkscreen used	LPI
Gold Fingers	Yes if there are gold fingers on the PCB, such as those found on the edge of a PCI card	NO
Blind vias	The existence of vias that go from one surface, and end in the middle of the PCB somewhere	Yes
Buried vias	The existence of vias that start and finish on layers other than the surface	No
PCB Dimensions	Dimensions of Square Envelope that the controller fits into. This is specified by IEEE p.1386 for a double wide CMC.	149mm x 149mm
CNC Route Points	Number of Corners / Radii to cut board	4
Layers	Number of layers in the PCB	8
Number of Via Pairs	Number of via pairs used in PCB	4
Number of Vias	Number of vias in design	1912
Number of Plated Hole Sizes	Number of plated hole sizes in PCB	5
Number of Nonplated hole Sizes	Number of non-plated hole sizes in PCB	3

9.2 VHDL Code

Table 16: Universal Controller VHDL File Descriptions

File Name	Description
FPGATOP.VHD	Top level FPGA file to combine modules
HEXTEST.UCF	Constraints file on pin locations and timing
COMMSTD.VHD	CommStack General Definitions
COMMSTACK.VHD	Top level PESNet 2.2 protocol stack
ASYNCHRONOUS_DM.VHD	Asynchronous Data Manager
PIPE_MUX16.VHD	Pipelined Multiplexer
CMDPROC.VHD	Command Processor
DDMTABLE.VHD	DSP Data Manager Transmitter Table Interface
ADDR_REG.VHD	Addressable Register
DDMRXTABLE.VHD	DSP Data Manager Receiver Table Interface
DDM_RX_FIFO_DP.VHD	DSP Data Manager packet FIFO
EVENT.VHD	Settable Event Vector
PIPE_COMPARE.VHD	Pipelined Comparison
DEFRAMER.VHD	Deframer (Packets to characters)
CRCGEN.VHD	Cyclic Redundancy Check Generator
EVENT_MAN.VHD	Event Manager
EXTENDED_MGR.VHD	Extended Data Manager
FRAMER.VHDL	Framer (Characters to packets)
INDEXMGR.VHD	Index Manager
WATCHDOG.VHD	Packet Watchdog
NETCLK_MGR.VHD	Net Clock Manager
NORMAL_DM.VHD	Normal Data Manager
REDIRECTOR.VHD	Redirector
SCHED_DM.VHD	Scheduled Data Manager
CONTROLREG.VHD	Control Registers (FPGA)
DAC.VHD	Digital to Analog Converter control
DBUSMUX.VHD	DSP Data Bus manager
DFLASH.VHD	DSP Flash Controller
DIPSW.VHD	DIP Switch Controller
DSP.VHD	DSP Reset Controller
HEX.VHD	Hexadecimal Display Controller
PBUSCTL.VHD	Peripheral Bus Controller
PLACEHOLDER.VHD	Temporary Placeholder for future modules
PMC.VHD	PCI Mezzanine Card Controller
RA_PWM.VHD	Right-Aligned PWM Module (for test purposes)
SELECTOR.VHD	Selector (Function Selector in FPGA servicing DSP)

VHDL File Structure:



9.2.1 FPGATop.VHD

```
------------------------------------------------------
------------------------------------------------------
-- HEADER
--   Project      : Bettis/Virginia Tech Application
Manager FPGA
--   Block        : FPGATOP
--   File Name    : FPGATOP.vhd
------------------------------------------------------
-- DESCRIPTION
--   Title        : PMC (Dual Port RAM Interface)
--   This block is the top level entity.  It combines
all other blocks and manages
--   some combinational signals.
------------------------------------------------------
-- REVISION HISTORY
--   Date        Author         Rev   Comments
--   03-27-02    J. Francis     -     Original design.
------------------------------------------------------
-- SYNTHESIS OPTIONS (or Synthesis Control File)
--   Control File    : None
--   Synthesis Tool  : Xilinx Foundation ISE 6.3
------------------------------------------------------
------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;
--
entity hextest is
    Port (

   HSCLK: in std_logic;  --high speed clock (80 MHz);

PDATA: inout std_logic_vector( 31 downto 0 );     --
PADDR: inout std_logic_vector( 18 downto 0 );

HEXBLANK: out std_logic;                    --W5
HEXLATCH: out std_logic;                    --V4

--Dsp clock enable - 0 = on
DSPCLKEN_L: out std_logic;                  --E17

--DSP Clock
DSPCLK: in std_logic;                       --AL17

--Reset signal for 2.5 Volt supervisor
RESET_L: in std_logic;                      --V5

--PMC Busmode (per CMC parent spec)
BUSMODE_L: inout std_logic_vector( 4 downto 1 );

JPERIPHD: out std_logic_vector( 48 downto 1 );--AJ16

PFLASHCS: out std_logic;                    --R5
PFLASHRESET: out std_logic;                 --R1

PCISELECT: out std_logic;                   --N4
PCISTROBE: out std_logic;
PREAD: out std_logic;
PWRITE: out std_logic;
PBLAST: out std_logic;
PCIALE: out std_logic;
PCIRDY_IN: out std_logic;
PCICLKIN: in std_logic;
PCIIRQ_IN: out std_logic;
PCITEST: out std_logic;
PCISDA: in std_logic;  --disable for now.
PCISCL: in std_logic;

PCIRDY_OUT: in std_logic;
PCIRSTOUTD: in std_logic;
PCIRSTOUT_L: in std_logic;
 PCIIRQ_OUT: in std_logic;
```

```
PCIPCLKOUT: in std_logic;

FORCLK: in std_logic;                --RCLK  AJ18

DACRST: out std_logic;                       --V3
DACCS: out std_logic;                        --V2
DACLDA: out std_logic;                       --W3
DACLDB: out std_logic;                       --W1
DACMSB: out std_logic;                       --U2
DACAB: out std_logic;                        --U4

PERIPHA: out std_logic_vector( 39 downto 1 );

DIPTRST: out std_logic;                      --U1
DFLASHCE_L: out std_logic;                   --W31
DSPRESET_L: out std_logic;                   --K31

ID:  in std_logic_vector( 2 downto 0 );
EBOOT: in std_logic;

RPBA: in std_logic;

IRQ1_L: inout std_logic;
IRQ2_L: inout std_logic;
IRQ0_L: inout std_logic;

FLAG: out std_logic_Vector( 3 downto 0 );
DSPTIMEXP_L: in std_logic;

PA_L: in std_logic;
SBTS_L: in std_logic;

TCK1: in std_logic;
TFS1: in std_logic;
DT1: in std_logic;
RCLK1: out std_logic;
RFS1: out stD_logic;
DR1: out std_logic;

DATA: inout std_logic_vector( 31 downto 0 );
ADDR: in std_logic_vector( 31 downto 0 );

   RDL_L: in std_logic;
   RDH_L: in std_logic;
   WRL_L: in std_logic;
   WRH_L: in stD_logic;

--  MS_L: in std_logic_vector( 3 downto 0 );

   BMS_L: in stD_logic;
   BRST: in stD_logic;
   PAGE: in stD_logic;
   ACK: inout std_logic;

   DMAR1_L: out std_logic;
   DMAG1_L: in std_logic;

   CS_L: out std_logic;
   HBR_L: out std_logic;
   HBG_L: in std_logic;

   REDY: in std_logic;


   DFLASHWE_L: out stD_logic;
   DFLASHOE_L: out std_logic;

--Multiple board interface
   MPCLK: inout std_logic;
   MPD: inout std_logic_vector( 6 downto 1);

--Upper level control interface
   ULCTL: inout std_logic_vector( 10 downto 1 );

-- FIBER OPTIC SIGNALS
   FO1LFI: in std_logic;
   FO1RESET: out std_logic;
   FO1RXCLK: in std_logic;
   FO1RXCMD: in std_logic_vector( 3 downto 0 );
   FO1RXDATA: in std_logic_vector( 7 downto 0 );
   FO1RXEMPTY: in std_logic;
   FO1RXSCD: in std_logic;
   FO1TXCMD: out std_logic_vector(3 downto 0);
```

```
    FO1TXDATA: out std_logic_vector( 7 downto 0);
    FO1TXEMPTY: in std_logic;
    FO1TXHALT: out std_logic;
    FO1TXSCD: out std_logic;
    FO1VLTN: in std_logic;

--  FIBER OPTIC SIGNALS
    FO2LFI: in std_logic;
    FO2RESET: out std_logic;
    FO2RXCLK: in std_logic;
    FO2RXCMD: in std_logic_vector( 3 downto 0 );
    FO2RXDATA: in std_logic_vector( 7 downto 0 );
    FO2RXEMPTY: in std_logic;
    FO2RXSCD: in std_logic;
    FO2TXFULL: in std_logic;
    FO2TXCMD: out std_logic_vector(3 downto 0);
    FO2TXDATA: out std_logic_vector( 7 downto 0);
    FO2TXEMPTY: in std_logic;
    FO2TXHALT: out std_logic;
    FO2TXSCD: out std_logic;
    FO2VLTN: in std_logic


   );

end hextest;

architecture behavioral of hextest is
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++
  component commstack is
      Port (
        --Global inputs
       BLOCK_HSCLK: in std_logic;
       BLOCK_RESET_L: in std_logic;

       --Event manager outputs                                    );
       FAULT_INJ: out std_logic;                -- fault
injection for fault protection testing
       FAULT_SOURCE_L: in std_logic_vector (4 downto 0);
-- current fault sources
       FAULT_RESET_L: out std_logic_vector (4 downto 0);
-- resets appropriate faults
       --Synchronous data manager outputs
         ACTIVE_DATA_SYNC: out T_SYNC_ATTR_ARRAY;

       --Normal data manager signals
       STATUS_DATA: in T_NORMAL_DATA_ARRAY;
       ACTIVE_DATA_NORMAL: out T_NORMAL_DATA_ARRAY;

       OPEN_LOOP_MODE_INDEX: in std_logic;
       PKT_ALIGNMENT: in std_logic_vector( 4 downto 0 );

       --packet coming in from redirector
       VLTN: in std_logic;
       LFI_L: in std_logic;
       TXDATA1: out std_logic_vector( 7 downto 0);
       TXCMD1: out std_logic_vector( 3 downto 0 );
       TXSCD1: out std_logic;
       REFCLK1: in std_logic;
       TXFULL_L: in std_logic;
       RXDATA1: in std_logic_vector( 7 downto 0);
       RXCMD1: in std_logic_vector( 3 downto 0 );
       RXSCD1: in std_logic;
       RXCLK1: in std_logic;
       -- Environmental info
       MAS_ADDR:    in T_NODEADDR;
       PKT_LENGTH: in std_logic_vector( 4 downto 0 );
       SYS_RST_COMM: out std_logic;
       PIN_RST: in std_logic;

       PKT_MODE: in E_PACKET_MODE;

       --interface to DSP
       BLOCK_DATA_IN: in std_logic_vector( 31 downto 0);
       BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0);
       BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0
       BLOCK_WRITE_L: in std_logic;
       BLOCK_ENABLE: in std_logic;
       BLOCK_DONE: out std_logic;

       SYNC: out std_logic;
       NET_TIME_OUT: out std_logic_vector( 7 downto 0 );
```

```
       ASYNC_DATA_OUT: out T_ASYNC_ATTR_ARRAY;
       ASYNC_DATA_IN: in T_ASYNC_ATTR_ARRAY;
       ASYNC_DATA_DIR: in std_logic_vector(
C_ASYNC_DATA_MAX downto 0 );  -- 1 = data comes from
packet, 0 = device

       DEBUG_DEF: out std_logic_vector( 7 downto 0 );
       DEBUG_FRM: out std_logic_vector( 7 downto 0 );

       DEBUG_AUX: out std_logic_vector( 32 downto 1 )

    );
  end component;


--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component controlreg is
    Port (
   BLOCK_HSCLK: in std_logic;
   BLOCK_DONE: out std_logic;
   BLOCK_RESET_L: in stD_logic;
   BLOCK_ENABLE: in std_logic;
   BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
   BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );
   BLOCK_ADDR_IN: in std_logic_vector( 19 downto 0 );
   BLOCK_WRITE_L: in std_logic;

   -- DAC control/status signals
   DACMSB: out std_logic;
   DACRST: out std_logic;

   -- DSP control/status signals
   DSPSTAT: in std_logic_vector( 7 downto 0 );

   --Fiber optic control/status signals
   FO_CONTROL: out std_logic_vector( 1 downto 0 );
   FO_STATUS: in std_logic_vector( 7 downto 0 );

   FO2_FIBER_ADDR: out std_logic_vector( 15 downto 0 );

   FO2_PHASEA: out std_logic_vector( 31 downto 0 );
   FO2_PHASEB: out std_logic_vector( 31 downto 0 );
   FO2_PHASEC: out std_logic_vector( 31 downto 0 );

   FO2_SENSORA: in std_logic_vector( 31 downto 0 );
   FO2_SENSORB: in std_logic_vector( 31 downto 0 );
   FO2_SENSORC: in std_logic_vector( 31 downto 0 );


   --Hex display control/status signals
   HEXBLANK: out std_logic;
   UNUSED: in std_logic_vector( 31 downto 0 )
);
end component;


--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component dac is
   Port (
   -- Block reset for initialization
   BLOCK_RESET_L: in std_logic;

   -- DSP is selecting the peripherals
   BLOCK_ENABLE: in std_logic;

   -- Controls whether or not to read from device
   BLOCK_WRITE_L: in std_logic;

   -- Synchronization line indicating that the block is
working when it is low
   BLOCK_DONE: out std_logic;

   -- Address in lines for reading address from FPGA
   BLOCK_ADDR_IN: in std_logic_vector( 2 downto 0 );

   --high speed clock (160 MHz)
   BLOCK_HSCLK: in std_logic;


   WR_DAC: out std_logic;
   DACLDA: out std_logic;
   DACLDB: out std_logic;
   DACCS:  out std_logic;
   DACAB: out std_logic
```

```
);
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component dbusmux is
    Port (
    BLOCK_HSCLK: in std_logic;
    BLOCK_READ_L: in std_logic;
    SELECTED: in std_logic;
    SEL_REG: in std_logic;

    REG_IN: in std_logic_vector( 31 downto 0 );
    PBUS_IN: in std_logic_vector( 31 downto 0 );

    DATA_OUT: out std_logic_vector( 31 downto 0 );
    PESNET_IN: in std_logic_vector( 31 downto 0 );
    SEL_PESNET: std_logic

   );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component dflash is
    Port (
    -- Block reset for initialization
    BLOCK_RESET_L: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to read from device
    BLOCK_READ_L: in std_logic;

    -- Controls whether to write to device
    BLOCK_WRITE_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    --DFLASH signals
    DFLASHOE_L: out std_logic;
    DFLASHWE_L: out std_logic;
    DFLASHCE_L: out std_logic
   );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component dipsw is
    Port (
    -- Block reset for initialization
    BLOCK_RESET: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to read from device
    BLOCK_READ_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    -- Latch data before disabling peripheral device
    BLOCK_PDATA_LATCH: out std_logic;

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    DIPTRST: out std_logic
   );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component dsp is
    Port (
    BLOCK_DONE: out std_logic;
    BLOCK_ENABLE: in std_logic;
    BLOCK_HSCLK: in std_logic;
    BLOCK_RESET: in std_logic;
```

```
    DSPSTAT: out std_logic_vector( 7 downto 0 );

    DSPCLK: in std_logic;
    RESET_L: in std_logic;

    DSPCLKEN_L: out std_logic;
    DSPRESET_L: out std_logic;
    --K31

    BMS_L: in stD_logic;
    BRST: in stD_logic;
    CS_L: out std_logic;
    DMAR1_L: out std_logic;
    DMAG1_L: in std_logic;
    DR1: out std_logic;
    DT1: in std_logic;
    DSPTIMEXP_L: in std_logic;
    EBOOT: in std_logic;
    FLAG: out std_logic_Vector( 3 downto 0 );
    HBR_L: out std_logic;
    HBG_L: in std_logic;
--   IRQ1_L: inout std_logic;
    IRQ2_L: inout std_logic;
    IRQ0_L: inout std_logic;
    PA_L: in std_logic;
    PAGE: in stD_logic;
    RCLK1: out std_logic;
    REDY: in std_logic;
    RFS1: out stD_logic;
    RPBA: in std_logic;
    SBTS_L: in std_logic;
    TCK1: in std_logic;
    TFS1: in std_logic
  );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component FIBER1 is
     Port (
    BLOCK_HSCLK: in std_logic;
     BLOCK_RESET_L: in std_logic;

    --Fiber optic control and status registers
    CONTROL: in std_logic_vector( 1 downto 0 );
    STATUS: out std_logic_vector( 3 downto 0 );

     -- TAXIchip Control SIGNALS
    --Transmitter data sampled on rising edge of RCLK
    RCLK: in std_logic;

    RESET_L: out std_logic;


    TXSCD: out std_logic;


    --Indicates ready for another character
    TXFULL_L: in std_logic;
    TXCMD: out std_logic_vector(3 downto 0);
    TXDATA: out std_logic_vector( 7 downto 0);

    --When active, the transmitter will send sync
characters
    TXHALT_L: out std_logic;

    VLTN: in std_logic;

    --Link Fault Indication driven active whenever input
stream
    --is not suitable for data recovery
    LFI_L: in std_logic;

    RXDATA: in std_logic_vector( 7 downto 0 );
    RXCMD: in std_logic_vector( 3 downto 0 );

    RXSCD: in std_logic;

    RXEMPTY_L: in std_logic;


    --Toggles at character rate
    RXCLK: in std_logic;

    PHASEA: in std_logic_vector( 15 downto 0 );
```

```
    PHASEB: in std_logic_vector( 15 downto 0 );
    PHASEC: in std_logic_vector( 15 downto 0 );

    A_SENSOR: out std_logic_vector( 31 downto 0 );
    B_SENSOR: out std_logic_vector( 31 downto 0 );
    C_SENSOR: out std_logic_vector( 31 downto 0 );

    DSPREADY: in std_logic
  );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component fiber1_master is
    Port (
    BLOCK_RESET_L: in std_logic;
    BLOCK_HSCLK: in std_logic;

    SENSOR_PHA_1: out std_logic_vector( 15 downto 0);
    SENSOR_PHA_2: out std_logic_vector( 15 downto 0);
    SENSOR_PHB_1: out std_logic_vector( 15 downto 0);
    SENSOR_PHB_2: out std_logic_vector( 15 downto 0);
    SENSOR_PHC_1: out std_logic_vector( 15 downto 0);
    SENSOR_PHC_2: out std_logic_vector( 15 downto 0);

    PWM_PHA_1: in std_logic_vector( 15 downto 0);
    PWM_PHA_2: in std_logic_vector( 15 downto 0);
    PWM_PHB_1: in std_logic_vector( 15 downto 0);
    PWM_PHB_2: in std_logic_vector( 15 downto 0);
    PWM_PHC_1: in std_logic_vector( 15 downto 0);
    PWM_PHC_2: in std_logic_vector( 15 downto 0);

    DATA_VALID: in std_logic;
    ADDR_REG: in std_logic_vector( 15 downto 0 );

    PWM_SYNC: out std_logic;

    TXCLK: out std_logic;
    TXCMD: out std_logic_vector( 3 downto 0);
    TXDATA: out std_logic_vector( 7 downto 0 );
    TXHALT: out std_logic;
    TXSCD: out std_logic;
    TXEN: out std_logic;
    TXEMPTY: in std_logic;
    TXFULL_L: in std_logic;
    RXCMD: in std_logic_vector( 3 downto 0 );
    RXDATA: in std_logic_vector( 7 downto 0 );
    RXSCD: in std_logic;
    RXEMPTY_L: in std_logic;
    RXEN_L: out std_logic;
    RESET_L: out std_logic;
    RXCLK: in std_logic;
    LFI_L: in std_logic;
    VLTN: in std_logic;
    RCLK: in std_logic;
    DEBUG: out std_logic_vector( 7 downto 0)
  );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component hexdisp is
    Port (
    -- Block reset for initialization
    BLOCK_RESET: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to read from device
    BLOCK_WRITE_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    WR_HEX: out std_logic;

    HEXLATCH: out std_logic
);
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
component i2cprom is
    Port (
    BLOCK_RESET: in std_logic;
    BLOCK_HSCLK: in std_logic;

    PD_SDA: inout std_logic;
    PD_SCL: in std_logic
   );
end component;


--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component pbusctl1 is
    Port (
    BLOCK_HSCLK: in std_logic;

    --device enable from selector
    EN_DIPSW: in std_logic;
    EN_HEX: in std_logic;
    EN_PMC: in std_logic;
    EN_DAC: in std_logic;

    --Latch input data from peripheral bus
    LATCH_PMC: in std_logic;
    LATCH_DIPSW: in std_logic;

    PREAD_PMC: in std_logic;

    PWRITE_PMC: in std_logic;

    --address line controls
    ADDR_PMC: in std_logic_vector( 18 downto 0 );

    --output enables for data bus
    WR_PMC: in std_logic;
    WR_HEX: in std_logic;
    WR_DAC: in std_logic;


    --Signals going out to bus
    PREAD: out std_logic;
    PWRITE: out std_logic;
    PADDR: out std_logic_vector( 18 downto 0 );
    PDATA: inout std_logic_vector( 31 downto 0 );

    --Input from DSP
    DSP_DATA_IN: in std_logic_vector( 31 downto 0 );

    --Output to FPGA blocks
    PDATA_IN: out std_logic_vector( 31 downto 0 )
);
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component periph is
    Port (
    JPERIPHA: inout std_logic_vector( 39 downto 1 );
    JPERIPHD: inout std_logic_vector( 48 downto 1 );

    --Select for auxiliary source of perpha signals
    PASOURCE1: in std_logic_vector( 39 downto 1  );
    PDSOURCE1: in std_logic_vector( 48 downto 1 );

    PASOURCE2: in std_logic_vector( 39 downto 1  );
    PDSOURCE2: in std_logic_vector( 48 downto 1 );

    --Peripheral IO control registers from CTLREG
    PER_CONTROL: in std_logic_vector( 7 downto 0 );
    PER_D_OUTL: in std_logic_vector( 23 downto 0 );
    PER_D_OUTH: in std_logic_vector( 23 downto 0 );
    PER_D_DIRL: in std_logic_vector( 23 downto 0 );
    PER_D_DIRH: in std_logic_vector( 23 downto 0 );
    PER_D_INL:  out  std_logic_vector( 23 downto 0 );
    PER_D_INH:  out  std_logic_vector( 23 downto 0 );
    PER_A_OUTL: in std_logic_vector( 23 downto 0 );
    PER_A_OUTH: in std_logic_vector( 14 downto 0 );
    PER_A_DIRL: in std_logic_vector( 23 downto 0 );
    PER_A_DIRH: in std_logic_vector( 14 downto 0 );
    PER_A_INL:  out  std_logic_vector( 23 downto 0 );
    PER_A_INH:  out  std_logic_vector( 14 downto 0 )

   );
end component;
```

```
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component placeholder is
    Port (
   RESET_L: in std_logic;
   BLOCK_ENABLE: in std_logic;
   BLOCK_DONE: out std_logic;
   HSCLK: in std_logic
  );
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component pmc IS
  port (
        BLOCK_HSCLK: in std_logic; --PCI Clock (40MHz)
        BLOCK_ENABLE: in std_logic; --PMC Block Enable
        BLOCK_RESET_L: in std_logic; --PMC Block Reset
        BLOCK_DONE: out std_logic; --PMC Block Busy
        BLOCK_READ_L: in std_logic; --DSP rd low strobe
        BLOCK_WRITE_L: in std_logic; --DSP wr low strobe
        blast_l      : out std_logic; --DP Burst Last
        data_en      : out std_logic; --Enables data bus
        dp_rd_l      : out std_logic; --DP Read
        dp_wr_l      : out std_logic; --DP Write
        reg_en       : out std_logic;  --register latch
        PMCCLK       : in std_logic;
        rdy_out_l    : in std_logic; --Rdy from DP RAM
        ale          : out std_logic; --DP Address Latch
        rdy_in       : out std_logic; --DP Ready In
        select_l     : out std_logic; --DP Select
        strobe_l     : out std_logic; --DP Strobe
        test         : out std_logic); --DP Test
END component;



--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component REG is
    Port (
   RESET_L: in std_logic;
   BLOCK_ENABLE: in std_logic;
   BLOCK_DONE: out std_logic;
   BLOCK_READ: in std_logic;
   BLOCK_WRITE: in std_logic;

   BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
   BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );
   BLOCK_ADDR_IN: in std_logic_vector( 31 downto 0 );

   HSCLK: in std_logic;

   WRH: in std_logic;

   DEBUG: out std_logic_vector( 39 downto 1 )
  );
end component;

component RA_PWM is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  DUTY_COUNT: in std_logic_vector( 15 downto 0 );
  PWM_SAT: in std_logic_vector( 15 downto 0 );
  SYNC: in std_logic;

  PWM_OUT: out std_logic
);
end component;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component SELECTOR is
     Port (
     --High speed clock
   BLOCK_HSCLK: in std_logic;
   BLOCK_RESET_L: in std_logic;
   BLOCK_READ_L: in std_logic;
   BLOCK_WRITE_L: in std_logic;

   --FPGA ID
   ID: in std_logic_vector( 2 downto 0 );

   --DSP Address bus
```

```
    ADDR: in std_logic_vector( 31 downto 25 );
     ACK: inout std_logic;  --ACK is pull-up
    BMS_L: in std_logic;

    --Device select lines.  Only one of these should be
on.  These are high when the device is selected
    EN_DAC: out std_logic;
    EN_DIPSW: out stD_logic;
    EN_DSP: out std_logic;
    EN_FPGA: out stD_logic;
    EN_HEX: out std_logic;
    EN_PERIPH: out std_logic;
    EN_PEEPROM: out std_logic;
    EN_PESNET: out std_logic;
    EN_PMC: out std_logic;
    EN_ULCTL: out stD_logic;
    EN_DFLASH: out std_logic;
--NEW MODULE: Insert EN_MODULE here


    --These go low when the device is working
    DONE_DAC: in std_logic;
    DONE_DIPSW: in std_logic;
    DONE_DSP: in std_logic;
    DONE_FPGA: in std_logic;
    DONE_HEX: in std_logic;
    DONE_PERIPH: in stD_logic;
    DONE_PEEPROM: in stD_logic;
    DONE_PESNET: in stD_logic;
    DONE_PMC: in stD_logic;
    DONE_ULCTL: in std_logic;
    DONE_DFLASH: in std_logic;
--NEW MODULE: Insert DONE_MODULE here

    --These are used when the FPGA wants to talk to the
DSP first
    --Examples of this usage would be a request from PMC
or Fiber
    --When in host bus access mode, the selector ignores
its inputs until
    --the host bus request line goes low.

    --In order to get in this mode, the DSP block will
set HOST_BUS_ACCESS_REQUEST
    --to a 1, and then wait for HOST_BUS_ACCESS_GRANT to
go to a 1.  When it goes to a 1,
    --the rest of the FPGA can ensure that the selector
does not have access to the DSP bus. during this time,
    --the DSP control block can use the host bus grant
lines to access DSP internal memory and perform
    --transactions with the DSP.
    HOST_BUS_ACCESS_REQUEST: in std_logic;
    HOST_BUS_ACCESS_GRANT: out std_logic;

    SELECTED: out std_logic --When 1, then the FPGA is
addressed
   );
end component;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  signal BLOCK_READ: std_logic;
  signal BLOCK_WRITE: std_logic;

  signal DFLASHOE_LX: std_logic;
  signal DFLASHWE_LX: stD_logic;
  signal DFLASHCE_LX: std_logic;

  signal RESET_LX: std_logic;
  signal RESET_LX2: std_logic;

  --selector signals
  signal HOST_BUS_ACCESS_REQUEST: std_logic;
  signal HOST_BUS_ACCESS_GRANT: std_logic;

  signal EN_DAC:  std_logic;
  signal EN_DIPSW:  stD_logic;
  signal EN_DSP:  std_logic;
  signal EN_FPGA:  stD_logic;
  signal EN_HEX:  std_logic;
  signal EN_PERIPH:  std_logic;
  signal EN_PEEPROM:  std_logic;
  signal EN_PESNET:  std_logic;
  signal EN_PMC:  std_logic;
  signal EN_ULCTL:  stD_logic;
```

```
  signal EN_DFLASH: std_logic;

  signal DONE_DAC:  std_logic;
  signal DONE_DIPSW:  stD_logic;
  signal DONE_DSP:  std_logic;
  signal DONE_FPGA:  stD_logic;
  signal DONE_HEX:  std_logic;
  signal DONE_PERIPH:  std_logic;
  signal DONE_PEEPROM:  std_logic;
  signal DONE_PESNET:  std_logic;
  signal DONE_PMC:  std_logic;
  signal DONE_ULCTL:  stD_logic;
  signal DONE_DFLASH: std_logic;

  --Control/status register signals
  signal CREG_DATA_OUT: std_logic_vector( 31 downto 0 );

  signal FO_CONTROL:  std_logic_vector( 1 downto 0 );
  signal FO_STATUS:   std_logic_Vector( 7 downto 0 );

  --for pesnet 1.2
  signal FO2_FIBER_ADDR: std_logic_vector( 15 downto 0);
  signal FO2_PHASEA: std_logic_vector( 31 downto 0 );
  signal FO2_PHASEB: std_logic_vector( 31 downto 0 );
  signal FO2_PHASEC: std_logic_vector( 31 downto 0 );
  signal FO2_SENSORA: std_logic_vector( 31 downto 0 );
  signal FO2_SENSORB: std_logic_vector( 31 downto 0 );
  signal FO2_SENSORC: std_logic_vector( 31 downto 0 );

  signal FO1_SENSORA: std_logic_vector( 31 downto 0 );
  signal FO1_SENSORB: std_logic_vector( 31 downto 0 );
  signal FO1_SENSORC: std_logic_vector( 31 downto 0 );

  signal NET_TIME: std_logic_vector( 7 downto 0 );
  signal PKT_ALIGNMENT: std_logic_vector( 4 downto 0 );
  signal OPEN_LOOP_MODE_INDEX: std_logic;
  signal SYS_RST_COMM: std_logic;
  signal ASYNC_DATA_OUT: T_ASYNC_ATTR_ARRAY;
  signal ASYNC_DATA_IN:T_ASYNC_ATTR_ARRAY;
  signal ASYNC_DATA_DIR: std_logic_vector(
C_ASYNC_DATA_MAX downto 0 );
  signal PKT_LENGTH: std_logic_vector( 4 downto 0 );

  signal PIN_RST: std_logic;


  signal DSPSTAT: std_logic_vector( 7 downto 0 );

  signal PADDR_OUT_PMC: std_logic_vector( 18 downto 0 );
  signal PDATA_WR_PMC: std_logic;
  signal PDATA_RD_PMC: stD_logic;

  --PBUSCTL signals
  signal LATCH_DIPSW: std_logic;
  signal LATCH_PMC: std_logic;

  signal WR_HEX: std_logic;
  signal WR_DAC: std_logic;
  signal WR_PMC: std_logic;

  signal HEXLATCHX: std_logic;
  signal DIPTRSTX: std_logic;
  --other signals
  signal DATA_OUT: std_logic_vector( 31 downto 0 );

  signal PDATA_IN: std_logic_vector( 31 downto 0 );

 -- signal PWRITE_PMC: std_logic;

  signal PADDR_PMC: std_logic_vector( 18 downto 0 );

  signal SELECTED: std_logic;

 -- signal PD1TEST: std_logic_vector( 48 downto 1 );

  signal HSCLKX: std_logic;

  signal PCIALEX: std_logic;
  signal PCIRDY_INX: std_logic;
  signal PCISELECTX: std_logic;
  signal PCISTROBEX: std_logic;
  signal PCITESTX: std_logic;
  signal PCICLKINX: std_logic;
  signal PBLASTX: std_logic;
  signal ACKX: std_logic;
```

```
  signal PREADX, PWRITEX: std_logic;

  --Fiber mirror signals for internal feedback
  signal FO1TXCMDX: std_logic_vector( 3 downto 0 );
  signal FO1TXSCDX: std_logic;
  signal FO1TXDATAX: std_logic_vector( 7 downto 0 );

  signal SYNC: std_logic;
  signal PWM_OUT: std_logic;

  signal UNUSED: std_logic_vector( 31 downto 0 );
  signal FO2TXCMDX: std_logic_vector( 3 downto 0 );
  signal FODEBUG: std_logic_vector( 7 downto 0 );
  signal FO2_TXFULL_L: std_logic;

  --commstack signals
  signal FAULT_INJ: std_logic;
  signal FAULT_SOURCE_L: std_logic_vector( 4 downto 0);
  signal FAULT_RESET_L: std_logic_vector( 4 downto 0 );
  signal ACTIVE_DATA_SYNC: T_SYNC_ATTR_ARRAY;
  signal STATUS_DATA: T_NORMAL_DATA_ARRAY;
  signal ACTIVE_DATA_NORMAL: T_NORMAL_DATA_ARRAY;
  signal BLOCK_DATA_OUT_PESNET: std_logic_vector( 31
downto 0 );
  signal DEBUG_DEF: std_logic_vector( 7 downto 0 );
  signal DEBUG_FRM: std_logic_vector( 7 downto 0 );
  signal DEBUG_AUX: std_logic_vector( 32 downto 1);
  signal IRQ0_LX: std_logic;

begin

ULCTL <= "ZZZZZZZZZZ";
MPCLK <= 'Z';
MPD <= "ZZZZZZ";

--PCICLKIN <= 'Z'; --PCICLKINX;
PCIALE <= PCIALEX;
PCIRDY_IN <= PCIRDY_INX;
PCISELECT <= PCISELECTX;
PCISTROBE <= PCISTROBEX;
PCITEST <= PCITESTX;

PREAD <= PREADX;
PWRITE <= PWRITEX;
HSCLKX <= HSCLK;
HEXLATCH <= HEXLATCHX;
DIPTRST <= DIPTRSTX;
PBLAST <= PBLASTX;

--=======================================================
-- Unused peripheral / address space place holders
--(Prevents deadlock when addressing unused devices)
--=======================================================
--PLACEHOLDER_DAC:     PLACEHOLDER port map( RESET_L,
EN_DAC,   DONE_DAC,     DSPCLK );
--PLACEHOLDER_DDFLASH: PLACEHOLDER port map( RESET_L,
EN_DFLASH, DONE_DFLASH,    DSPCLK );
--PLACEHOLDER_DIPSW:   PLACEHOLDER port map( RESET_L,
EN_DIPSW,  DONE_DIPSW,  DSPCLK );
--PLACEHOLDER_DSP:     PLACEHOLDER port map( RESET_L,
EN_DSP,   DONE_DSP,     DSPCLK );
--PLACEHOLDER_FPGA:    PLACEHOLDER port map( RESET_L,
EN_FPGA,  DONE_FPGA,    DSPCLK );
--PLACEHOLDER_HEX:     PLACEHOLDER port map( RESET_L,
EN_HEX,   DONE_HEX,     DSPCLK );
PLACEHOLDER_PERIPH:    PLACEHOLDER port map( RESET_L,
EN_PERIPH, DONE_PERIPH, DSPCLK );
PLACEHOLDER_PEEPROM:   PLACEHOLDER port map( RESET_L,
EN_PEEPROM,  DONE_PEEPROM, HSCLK  );
--PLACEHOLDER_PESNET:  PLACEHOLDER port map( RESET_L,
EN_PESNET,   DONE_PESNET, DSPCLK );
--PLACEHOLDER_PMC:     PLACEHOLDER port map( RESET_L,
EN_PMC,   DONE_PMC,     DSPCLK );
PLACEHOLDER_ULCTL:     PLACEHOLDER port map( RESET_L,
EN_ULCTL, DONE_ULCTL,   DSPCLK );

--=======================================================


--=======================================================
-- PESNet 2.2 Interface
--=======================================================
FO1TXHALT <= '1'; --<= FO_CONTROL(0);
FO1RESET <= RESET_L; --FO_CONTROL(1);
```

```
-- Timing:    _________|---------[12.5 ns]-----[12.5 ns]
-- Signal:  RESET_L (ASYNC)  ==>  RESET_LX2 ==> RESET_LX
process( HSCLK ) is
begin
-- if( RESET_L = '0' ) then
--  RESET_LX <= '0';
--  RESET_LX2 <= '0';
  if( rising_edge( HSCLK ) ) then
    --for comm stack
    PIN_RST <= RESET_L;

    if( RESET_L = '0' or SYS_RST_COMM = '1') then
     RESET_LX <= '0';
     RESET_LX2 <= '0';
    else
     RESET_LX <= RESET_LX2;
     RESET_LX2 <= RESET_L;
    end if;
  end if;
end process;

CCOMMSTACK: COMMSTACK port map(
  --DSP Interface
  BLOCK_HSCLK        => HSCLK,
  BLOCK_RESET_L      => RESET_LX,
  BLOCK_ENABLE       => EN_PESNET,
  BLOCK_DONE     => DONE_PESNET,
  BLOCK_DATA_IN      => DATA,
  BLOCK_WRITE_L      => BLOCK_WRITE,
  BLOCK_ADDR_IN      => ADDR( 15 downto 0 ),
  BLOCK_DATA_OUT     => BLOCK_DATA_OUT_PESNET,
  TXFULL_L    => FO1TXEMPTY,
  --PESNet
  PKT_MODE       => LONG_PKT,
--  THIS_ADDR        => X"7F",
  MAS_ADDR       => X"3E",
  NET_TIME_OUT       => NET_TIME,
  VLTN      => FO1VLTN,
  LFI_L     => FO1LFI,

  FAULT_INJ      => FAULT_INJ,
  FAULT_SOURCE_L     => FAULT_SOURCE_L,
  FAULT_RESET_L      => FAULT_RESET_L,
  ACTIVE_DATA_SYNC   => ACTIVE_DATA_SYNC,
  STATUS_DATA    => STATUS_DATA,
  ACTIVE_DATA_NORMAL => ACTIVE_DATA_NORMAL,

  --Receiver side
  RXCLK1      => FO1RXCLK,
  RXCMD1      => FO1RXCMD,
  RXDATA1     => FO1RXDATA,
  RXSCD1    => FO1RXSCD,
  --FO1RXEMPTY,

  --Transmitter side
  REFCLK1     => FORCLK,
  TXCMD1      => FO1TXCMDx,
  TXDATA1     => FO1TXDATAx,
  TXSCD1    => FO1TXSCDx,
  SYNC      => SYNC,

  PKT_ALIGNMENT      => PKT_ALIGNMENT,
  OPEN_LOOP_MODE_INDEX => OPEN_LOOP_MODE_INDEX,
  SYS_RST_COMM       => SYS_RST_COMM,
  ASYNC_DATA_OUT     => ASYNC_DATA_OUT,
  ASYNC_DATA_IN      => ASYNC_DATA_IN,
  ASYNC_DATA_DIR     => ASYNC_DATA_DIR,
  PKT_LENGTH     => PKT_LENGTH,
  PIN_RST     => PIN_RST,

  --FO1RXEMPTY,
  DEBUG_DEF      => DEBUG_DEF,
  DEBUG_FRM      => DEBUG_FRM,
  DEBUG_AUX      => DEBUG_AUX
);

FO1TXCMD <= FO1TXCMDX;
FO1TXSCD <= FO1TXSCDX;
FO1TXDATA <= FO1TXDATAX;

STATUS_DATA(0) <= FO2_PHASEA(15 downto 0 );
STATUS_DATA(1) <= FO2_PHASEA(31 downto 16);
STATUS_DATA(2) <= FO2_PHASEB(15 downto 0 );
```

```
STATUS_DATA(3) <= FO2_PHASEB(31 downto 16);

FO2_SENSORA <= ACTIVE_DATA_SYNC(1) &
ACTIVE_DATA_SYNC(0);
FO2_SENSORB <= ACTIVE_DATA_NORMAL(3) &
ACTIVE_DATA_NORMAL(2);
FO2_SENSORC <= X"0000" & X"00" & FAULT_RESET_L &
FAULT_INJ & "01";

FO1_SENSORA <= ACTIVE_DATA_NORMAL(1) &
ACTIVE_DATA_NORMAL(0);
--FO1_SENSORB <= ACTIVE_DATA_SYNC(2) &
ACTIVE_DATA_SYNC(3);

FAULT_SOURCE_L <= "11111";

--=======================================================
-- Control Register Interface
--=======================================================
CCTRLREG: controlreg port map(
  BLOCK_HSCLK    => DSPCLK,
  BLOCK_DONE  => DONE_FPGA,
  BLOCK_RESET_L  => RESET_LX,
  BLOCK_ENABLE   => EN_FPGA,
  BLOCK_DATA_IN  => DATA,
  BLOCK_DATA_OUT => CREG_DATA_OUT,
  BLOCK_ADDR_IN  => ADDR(19 downto 0 ),
  BLOCK_WRITE_L  => BLOCK_WRITE,
  DACMSB    => DACMSB,
  DACRST    => DACRST,
  DSPSTAT   => DSPSTAT,
  FO_CONTROL  => FO_CONTROL,
  FO_STATUS   => FO_STATUS,
  FO2_FIBER_ADDR => FO2_FIBER_ADDR,
  FO2_PHASEA  => FO2_PHASEA,
  FO2_PHASEB  => FO2_PHASEB,
  FO2_PHASEC  => FO2_PHASEC,
  FO2_SENSORA    => FO2_SENSORA,
  FO2_SENSORB    => FO2_SENSORB,
  FO2_SENSORC    => FO2_SENSORC,
  HEXBLANK    => HEXBLANK,
  UNUSED    => UNUSED
  );

--=======================================================
-- Digital to Analog Converter Controller
--=======================================================
CDAC: dac port map(
  BLOCK_ADDR_IN  => ADDR( 2 downto 0),
  BLOCK_DONE  => DONE_DAC,
  BLOCK_ENABLE   => EN_DAC,
  BLOCK_HSCLK    => DSPCLK,
  BLOCK_WRITE_L  => BLOCK_WRITE,
  BLOCK_RESET_L  => RESET_LX,

  WR_DAC    => WR_DAC,
  DACLDA    => DACLDA,
  DACLDB    => DACLDB,
  DACCS    => DACCS,
  DACAB    => DACAB
);

--=======================================================
-- DSP Bus Controller
--=======================================================
CDBUS: dbusmux port map(
  BLOCK_HSCLK    => HSCLKX,
  BLOCK_READ_L   => BLOCK_READ,
  SELECTED    => SELECTED,
  SEL_REG   => EN_FPGA,
  REG_IN    => CREG_DATA_OUT,
  PESNET_IN   => BLOCK_DATA_OUT_PESNET,
  PBUS_IN   => PDATA_IN,
  DATA_OUT    => DATA ,
  SEL_PESNET  => EN_PESNET
);

--=======================================================
-- DSP Flash Interface
--=======================================================
CDFLASH: dflash port map(
  BLOCK_DONE  => DONE_DFLASH,
  BLOCK_ENABLE   => EN_DFLASH,
  BLOCK_RESET_L  => RESET_LX,
```

```
  BLOCK_READ_L   => BLOCK_READ,
  BLOCK_WRITE_L  => BLOCK_WRITE,
  BLOCK_HSCLK => HSCLKX,
  DFLASHOE_L  => DFLASHOE_LX,
  DFLASHWE_L  => DFLASHWE_LX,
  DFLASHCE_L  => DFLASHCE_LX
);

DFLASHOE_L <= DFLASHOE_LX;
DFLASHWE_L <= DFLASHWE_LX;
DFLASHCE_L <= DFLASHCE_LX;

--=======================================================
-- DIP Switch Interface
--=======================================================
CDIPSW: DIPSW port  map(
  BLOCK_RESET => RESET_LX,
  BLOCK_DONE => DONE_DIPSW,
  BLOCK_ENABLE => EN_DIPSW,
  BLOCK_HSCLK => DSPCLK,
  BLOCK_READ_L => BLOCK_READ,
  BLOCK_PDATA_LATCH => LATCH_DIPSW,
  DIPTRST => DIPTRSTX
);


--=======================================================
-- DSP Support Interface
--=======================================================
DSPa: DSP port map(
  BLOCK_DONE => DONE_DSP,
  BLOCK_ENABLE => EN_DSP,
  BLOCK_HSCLK => HSCLK,
  BLOCK_RESET => RESET_LX,

  DSPSTAT => DSPSTAT,

  DSPCLK => HSCLKX,
  RESET_L => RESET_LX,

  DSPCLKEN_L => DSPCLKEN_L,
  DSPRESET_L => DSPRESET_L,
  BMS_L => BMS_L,
  BRST => BRST,
  CS_L => CS_L,
  DMAR1_L => DMAR1_L,
  DMAG1_L => DMAG1_L,
  DR1 => DR1,
  DT1 => DT1,
  DSPTIMEXP_L => DSPTIMEXP_L,
  EBOOT => EBOOT,
  FLAG => FLAG,
  HBR_L => HBR_L,
  HBG_L => HBG_L,
--  IRQ1_L => IRQ1_L,
  IRQ2_L => IRQ2_L,
--  IRQ0_L => IRQ0_L,
  PA_L => PA_L,
  PAGE => PAGE,
  RCLK1 => RCLK1,
  REDY => REDY,
  RFS1 => RFS1,
  RPBA => RPBA,
  SBTS_L => SBTS_L,
  TCK1 => TCK1,
  TFS1 => TFS1
);
--
=========================================================
=========================

FO1_SENSORB(15 downto 0) <= X"000" & FO2TXFULL & FO2VLTN
& FO2LFI & FO2RXEMPTY;
FO1_SENSORB(31 downto 16 ) <= X"0000";
FO1_SENSORC <= X"00000000";

FO2TXSCD <= FO2RXSCD;
FO2TXCMD <= FO2RXCMD;
FO2TXDATA <= FO2RXDATA;
FO2TXHALT <= FO_CONTROL(0);

FO2RESET <= FO_CONTROL(1);
IRQ1_L <= '1';
```

```
--=======================================================
-- Hex display controller
=======================================================
HEXDISPC: HEXDISP port map(
  BLOCK_DONE  => DONE_HEX,
  BLOCK_ENABLE   => EN_HEX,
  BLOCK_HSCLK    => DSPCLK,
  BLOCK_WRITE_L  => BLOCK_WRITE,
  BLOCK_RESET    => RESET_LX,
  WR_HEX   => WR_HEX,
  HEXLATCH    => HEXLATCHX
);

--=======================================================
-- Peripheral Bus Controller
--=======================================================
CPBUSCTL: PBUSCTL port map(
  BLOCK_HSCLK => HSCLK,

  EN_DIPSW    => EN_DIPSW,
  EN_HEX    => EN_HEX,
  EN_PMC    => EN_PMC,
  EN_DAC    => EN_DAC,

  LATCH_PMC   => LATCH_PMC, --LATCH_PMC,
  LATCH_DIPSW    => LATCH_DIPSW,

  PREAD_PMC   => PDATA_RD_PMC,
  PWRITE_PMC  => PDATA_WR_PMC,
  ADDR_PMC    => ADDR(18 downto 0 ),
  WR_DAC    => WR_DAC,
  WR_HEX    => WR_HEX,
  WR_PMC    => WR_PMC,

  PREAD    => PREADX,
  PWRITE    => PWRITEX,
  PADDR    => PADDR,
  PDATA    => PDATA,
  DSP_DATA_IN    => DATA,
  PDATA_IN    => PDATA_IN
);


  process( RESET_LX, HSCLK ) is
  begin
    if( rising_edge( HSCLK ) ) then
     if( RESET_LX = '0' ) then
        PCICLKINX <= '0';
     else
        PCICLKINX <= not PCICLKINX;
     end if;
    end if;
  end process;

  BUSMODE_L(4 downto 2 ) <= "ZZZ";
  BUSMODE_L(1) <= '0';


process( RESET_LX, HSCLK ) is
variable EXECUTED: std_logic;
begin
  if( rising_Edge( HSCLK ) ) then

    if( RESET_LX = '0' ) then
     FO2_TXFULL_L <= '0';
     EXECUTED := '0';
    else
     if( FORCLK = '1' and EXECUTED = '0') then
        EXECUTED := '1';
        FO2_TXFULL_L <= not FO2_TXFULL_L;
     else
        EXECUTED := '0';
     end if;
    end if;
  end if;
end process;

--=======================================================
-- PCI Mezzanine Card Interface
--=======================================================
```

```
CPMC: PMC port map(
  BLOCK_HSCLK    => HSCLK    , --High speed Clock (80MHz)
  BLOCK_ENABLE   => EN_PMC   , --PMC Block Enable
  BLOCK_DONE  => DONE_PMC,
  BLOCK_RESET_L  => RESET_LX,       --PMC Block Reset
  BLOCK_READ_L   => BLOCK_READ,
  BLOCK_WRITE_L  => BLOCK_WRITE,
  PMCCLK    => PCICLKIN,
  rdy_out_l   => PCIRDY_OUT , --Ready Out signal from DP
RAM
  dp_wr_l   => PDATA_WR_PMC,
  dp_rd_l   => PDATA_RD_PMC,
  data_en   => WR_PMC,
  ale         => PCIALEX    , --DP Address Latch Enable
  blast_l     => PBLASTX    , --DP Burst Last
  rdy_in      => PCIRDY_INX   , --DP Ready In
  select_l    => PCISELECTX   , --DP Select
  strobe_l    => PCISTROBEX   , --DP Strobe
  reg_en      => LATCH_PMC,
  test        => PCITESTX
  ); --DP Test


  PADDR_OUT_PMC( 18 downto 17 ) <= "00";
  PADDR_OUT_PMC(16 downto 0 ) <= ADDR(16 downto 0 );
  PCIIRQ_IN <= '1';

--=======================================================
-- Selector
--=======================================================
SELECTOR1: SELECTOR port map(
  BLOCK_HSCLK => HSCLKX,
  BLOCK_RESET_L => RESET_LX,
  ID => ID,
  ADDR => ADDR(31 downto 25),
  ACK => ACKX,
  BMS_L => BMS_L,

  EN_DAC => EN_DAC,
  EN_DIPSW => EN_DIPSW,
  EN_DSP => EN_DSP,
  EN_FPGA => EN_FPGA,
  EN_HEX => EN_HEX,
  EN_PERIPH => EN_PERIPH,
  EN_PEEPROM => EN_PEEPROM,
  EN_PMC => EN_PMC,
  EN_PESNET => EN_PESNET,
  EN_ULCTL => EN_ULCTL,
  EN_DFLASH => EN_DFLASH,

  DONE_DAC => DONE_DAC,
  DONE_DIPSW => DONE_DIPSW,
  DONE_DSP => DONE_DSP,

  DONE_FPGA => DONE_FPGA,
  DONE_HEX => DONE_HEX,
  DONE_PERIPH => DONE_PERIPH,
  DONE_PEEPROM => DONE_PEEPROM,
  DONE_PESNET => DONE_PESNET,
  DONE_PMC => DONE_PMC,
  DONE_ULCTL => DONE_ULCTL,
  DONE_DFLASH => DONE_DFLASH,

  BLOCK_READ_L => BLOCK_READ,
  BLOCK_WRITE_L => BLOCK_WRITE,
  SELECTED => SELECTED,
  HOST_BUS_ACCESS_REQUEST => HOST_BUS_ACCESS_REQUEST
);

--Process to filter glitches from ACK.  ACK will fall
ASAP, and rise on next rising_edge
process( HSCLK, RESET_LX, ACKX ) is
begin
-- if( ACKX = '0' ) then
--  ACK <= '0';
  if( rising_edge( HSCLK ) ) then
    if( ACKX = '0' ) then
     ACK <= '0';
    else
     ACK <= '1';
    end if;
  end if;
end process;
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
--=======================================================
--Consolidate writes and reads into one line
BLOCK_WRITE <= WRL_L and WRH_L; --and WRH_L;
BLOCK_READ <= RDL_L and RDH_L; -- and RDH_L;

HOST_BUS_ACCESS_REQUEST <= '0';

  --Disable PFLASH
  PFLASHCS <= '1';
  PFLASHRESET <= '0';


C_RA_PWM: RA_PWM port map(
  BLOCK_HSCLK => HSCLK,
  BLOCK_RESET_L => RESET_L,

  DUTY_COUNT => ACTIVE_DATA_NORMAL(2),
  PWM_SAT   => ACTIVE_DATA_NORMAL(3),
  SYNC    => SYNC,
  PWM_OUT   => PWM_OUT

);

process( HSCLK ) is
begin
  if( rising_edge( HSCLK ) ) then
  if( RESET_L = '0' ) then
    IRQ0_LX <= '1';
    IRQ0_L <= '0';
  else
    IRQ0_L <= IRQ0_LX;
    if( NET_TIME = FO2_FIBER_ADDR( 7 downto 0 ) ) then
     if( FO_CONTROL(0) = '1' ) then
        IRQ0_LX <= '0';
     else
        IRQ0_LX <= '0';  --for now also enable it
regardless
     end if;
    else
     IRQ0_LX <= '1';
    end if;
  end if;
  end if;
end process;


--=======================================================
-- Debug and Expansion connector interface
--=======================================================
  process( HSCLKX, RESET_LX) is
  variable M : std_logic;
  begin
    if( rising_edge( HSCLKX ) ) then
     if( RESET_LX = '0' ) then
        M := '0';
     else
        M := not M;
     end if;
    end if;
  end process;

     JPERIPHD(48 downto 41) <= DEBUG_FRM;
     JPERIPHD(40 downto 33) <= DEBUG_DEF;
        JPERIPHD(32 downto 1 ) <= DEBUG_AUX;

--=======================================================
  --This section is used to control PERIPHA, the 50 pin
connector designated J3

  process( RESET_LX, HSCLK) is
  variable M : std_logic;
  begin
  -- if( RESET_LX = '0' ) then
  --   M := '0';
    if( rising_edge( HSCLK ) ) then
     if( RESET_LX = '0' ) then
        M := '0';
     else
        M := not M;
        PERIPHA(29) <= M;
     end if;
     end if;
  end process;
```

```
    PERIPHA( 39 downto 36 ) <= FO1TXCMDX;
    PERIPHA(35) <= ACK;
    PERIPHA(34) <= RDL_L;
    PERIPHA(33) <= WRL_L;
    PERIPHA(32) <= FO1TXSCDX;
    PERIPHA(31) <= WR_DAC;
    PERIPHA(30) <= FAULT_INJ;
    PERIPHA(28) <= EN_PESNET;
    PERIPHA(27 downto 20 ) <= FO1TXDATAX;
    PERIPHA(19) <= PWRITEX;
    PERIPHA(18) <= '1';
    PERIPHA(17) <= FORCLK;
    PERIPHA(16 downto 13) <= FO1RXCMD;
    PERIPHA(12) <= SYNC;
    PERIPHA(11) <= FO1RXSCD;
    PERIPHA(10) <= FO1LFI;
    PERIPHA(9) <= FO1VLTN;
    PERIPHA(8 downto 1) <= FO1RXDATA;

--  end if;
-- end process;
  --


--================================================
ASYNC_DATA_IN(0)  <= ASYNC_DATA_DIR;
ASYNC_DATA_IN(1)  <= X"000"& '0' & ID;
ASYNC_DATA_IN(2)  <= OPEN_LOOP_MODE_INDEX & "00" &
PKT_ALIGNMENT & "000" & PKT_LENGTH;
ASYNC_DATA_IN(3)  <= FO2_FIBER_ADDR;
ASYNC_DATA_IN(4)  <= X"2345";
ASYNC_DATA_IN(5)  <= X"5678";
ASYNC_DATA_IN(6)  <= X"0006";    --line contactor status
word
ASYNC_DATA_IN(7)  <= X"0007";    --
ASYNC_DATA_IN(8)  <= X"0008";    --position code
ASYNC_DATA_IN(9)  <= X"0009";    --device temperature
ASYNC_DATA_IN(10) <= X"000A";    --master address
ASYNC_DATA_IN(11) <= X"000B";    --device class
ASYNC_DATA_IN(12) <= X"1012";    --hardware id
ASYNC_DATA_IN(13) <= X"0001";    --manuf id
ASYNC_DATA_IN(14) <= X"000E";    --hw revision
ASYNC_DATA_IN(15) <= X"0100";    --fw revision



--Control packet parameters
process( HSCLK ) is
begin
  if( rising_edge(HSCLK ) ) then
    if( RESET_LX = '0' ) then
     PKT_LENGTH <= CONV_STD_LOGIC_VECTOR( 19, 5);
     OPEN_LOOP_MODE_INDEX <= '0';
     PKT_ALIGNMENT <= CONV_STD_LOGIC_VECTOR( 17, 5 );

     ASYNC_DATA_DIR <= "0000000000000001";
    else
     PKT_LENGTH <= ASYNC_DATA_OUT(2)(4 downto 0 );
     OPEN_LOOP_MODE_INDEX <= ASYNC_DATA_OUT(2)(15);
     PKT_ALIGNMENT <= ASYNC_DATA_OUT(2)( 12 downto 8 );

     ASYNC_DATA_DIR <= ASYNC_DATA_OUT(0) or
"0000000000000001"; --allways enable control of
direction
    end if;
  end if;
end process;
--=========================================================


--=========================================================
-- Unused pins
--=========================================================
  --These signals may be unused.  They allow the
translate step to run.
  --These signals are readable as a register in the DSP.
  process( DSPCLK, FO2RXCLK ) is
  begin
    if( rising_edge( FO2RXCLK ) ) then
     if( FO2RXDATA = X"00" ) then
       UNUSED(31) <= '1'; --BUSMODE_L(4);
     else
       UNUSED(31) <=  FO2RXSCD;
     end if;
```

```
    if( FO2RXCMD = X"0" ) then
      UNUSED(30) <= BUSMODE_L(3);
    end if;
    UNUSED(29) <= BUSMODE_L(2);
    --UNUSED(28 downto 21) <= DSPSTAT;
    UNUSED( 28 downto 25) <= FO1_SENSORA(31 downto 28)
or FO1_SENSORA(27 downto 24);
    UNUSED(24) <= EBOOT;
    UNUSED(23) <= BUSMODE_L(1);
    UNUSED(22) <= PCISCL or PCISDA;
    UNUSED(21) <= PCIRDY_OUT;
    UNUSED(20) <= PCIRSTOUTD;
    UNUSED(19) <= PCIRSTOUT_L;
    UNUSED(18) <= PCIIRQ_OUT;
    UNUSED(17) <= PCIPCLKOUT;
    UNUSED(16) <= HBG_L or FORCLK;
    UNUSED(15) <= REDY or FO1RXEMPTY;
    UNUSED(14) <= BRST or FO1VLTN;
    UNUSED(13) <= PAGE or FO2VLTN;
    UNUSED(12) <= FO2LFI;
    UNUSED(11) <= FO2RXEMPTY;
    UNUSED(10) <= FO2TXEMPTY;
    UNUSED(9) <= RDH_L;
    UNUSED(8) <= FO2TXFULL;
    UNUSED(7 downto 2) <= ADDR( 24 downto 19 );
    UNUSED(1) <= RPBA or FO1TXEMPTY;
    UNUSED(0) <= WRH_L;
   end if;
  end process;

end behavioral;
```

9.2.2 HexTest.UCF

```
#### UCF FILE CREATED BY PROJECT NAVIGATOR
NET "DSPCLK" TNM_NET = "DSPCLK";
TIMESPEC "TS_DSPCLK" = PERIOD "DSPCLK" 42.5 MHZ HIGH 50
%;
NET "FO1RXCLK" TNM_NET = "FO1RXCLK";
TIMESPEC "TS_FO1RXCLK" = PERIOD "FO1RXCLK" 25 MHZ HIGH
50 %;
NET "HSCLK" TNM_NET = "HSCLK";
TIMESPEC "TS_HSCLK" = PERIOD "HSCLK" 85 MHZ HIGH 50 %;
TEMPERATURE = 40 ;
VOLTAGE = 2.45 ;
NET "PDATA<0>" PULLUP;
NET "PDATA<1>" PULLUP;
NET "PDATA<2>" PULLUP;
NET "PDATA<3>" PULLUP;
NET "PDATA<4>" PULLUP;
NET "PDATA<5>" PULLUP;
NET "PDATA<6>" PULLUP;
NET "PDATA<7>" PULLUP;
NET "PDATA<8>" PULLUP;
NET "PDATA<9>" PULLUP;
NET "PDATA<10>" PULLUP;
NET "PDATA<11>" PULLUP;
NET "PDATA<12>" PULLUP;
NET "PDATA<13>" PULLUP;
NET "PDATA<14>" PULLUP;
NET "PDATA<15>" PULLUP;
NET "PDATA<16>" PULLUP;
NET "PDATA<17>" PULLUP;
NET "PDATA<18>" PULLUP;
NET "PDATA<19>" PULLUP;
NET "PDATA<20>" PULLUP;
NET "PDATA<21>" PULLUP;
NET "PDATA<22>" PULLUP;
NET "PDATA<23>" PULLUP;
NET "PDATA<24>" PULLUP;
NET "PDATA<25>" PULLUP;
NET "PDATA<26>" PULLUP;
NET "PDATA<27>" PULLUP;
NET "PDATA<28>" PULLUP;
NET "PDATA<29>" PULLUP;
NET "PDATA<30>" PULLUP;
NET "PDATA<31>" PULLUP;
#NET "PCIIRQ_OUT" PULLUP;
#NET "PCIRDY_IN" PULLUP;
NET "PADDR<0>" PULLUP;
NET "PADDR<1>" PULLUP;
NET "PADDR<2>" PULLUP;
NET "PADDR<3>" PULLUP;
NET "PADDR<4>" PULLUP;
NET "PADDR<5>" PULLUP;
NET "PADDR<6>" PULLUP;
NET "PADDR<7>" PULLUP;
NET "PADDR<8>" PULLUP;
NET "PADDR<9>" PULLUP;
NET "PADDR<10>" PULLUP;
NET "PADDR<11>" PULLUP;
NET "PADDR<12>" PULLUP;
NET "PADDR<13>" PULLUP;
NET "PADDR<14>" PULLUP;
NET "PADDR<15>" PULLUP;
NET "PADDR<16>" PULLUP;
NET "PADDR<17>" PULLUP;
NET "PADDR<18>" PULLUP;
#PACE: START OF CONSTRAINTS EXTRACTED BY PACE FROM THE
DESIGN
NET "WRL_L" LOC = "U29";
NET "WRH_L" LOC = "T30";
NET "ULCTL<10>" LOC = "M2";
NET "ULCTL<9>" LOC = "N2";
NET "ULCTL<8>" LOC = "L3";
NET "ULCTL<7>" LOC = "J2";
NET "ULCTL<6>" LOC = "G1";
NET "ULCTL<5>" LOC = "K2";
NET "ULCTL<4>" LOC = "L1";
NET "ULCTL<3>" LOC = "G3";
NET "ULCTL<2>" LOC = "H2";
NET "ULCTL<1>" LOC = "J3";
```

```
NET "TFS1" LOC = "A19";
NET "TCK1" LOC = "G31";
NET "SBTS_L" LOC = "U31";
NET "RPBA" LOC = "E31";
NET "RFS1" LOC = "D18";
NET "RESET_L" LOC = "AH5";
NET "REDY" LOC = "P32";
NET "RDL_L" LOC = "R29";
NET "RDH_L" LOC = "T29";
NET "RCLK1" LOC = "E18";
NET "PWRITE" LOC = "C13";
NET "PREAD" LOC = "A13";
NET "PFLASHRESET" LOC = "R1";
NET "PFLASHCS" LOC = "R5";
NET "PERIPHA<39>" LOC = "AJ1";
NET "PERIPHA<38>" LOC = "AK2";
NET "PERIPHA<37>" LOC = "AK3";
NET "PERIPHA<36>" LOC = "AL1";
NET "PERIPHA<35>" LOC = "AM4";
NET "PERIPHA<34>" LOC = "AL4";
NET "PERIPHA<33>" LOC = "AM5";
NET "PERIPHA<32>" LOC = "AL5";
NET "PERIPHA<31>" LOC = "AN6";
NET "PERIPHA<30>" LOC = "AM6";
NET "PERIPHA<29>" LOC = "AL6";
NET "PERIPHA<28>" LOC = "AL7";
NET "PERIPHA<27>" LOC = "AN7";
NET "PERIPHA<26>" LOC = "AM8";
NET "PERIPHA<25>" LOC = "AL8";
NET "PERIPHA<24>" LOC = "AN9";
NET "PERIPHA<23>" LOC = "AM9";
NET "PERIPHA<22>" LOC = "AM10";
NET "PERIPHA<21>" LOC = "AL10";
NET "PERIPHA<20>" LOC = "AN11";
NET "PERIPHA<19>" LOC = "AL11";
NET "PERIPHA<18>" LOC = "AM12";
NET "PERIPHA<17>" LOC = "AL12";
NET "PERIPHA<16>" LOC = "AN13";
NET "PERIPHA<15>" LOC = "AM13";
NET "PERIPHA<14>" LOC = "AL13";
NET "PERIPHA<13>" LOC = "AM14";
NET "PERIPHA<12>" LOC = "AN15";
NET "PERIPHA<11>" LOC = "AL15";
NET "PERIPHA<10>" LOC = "AM16";
NET "PERIPHA<9>" LOC = "AL16";
NET "PERIPHA<8>" LOC = "AN17";
NET "PERIPHA<7>" LOC = "AM17";
NET "PERIPHA<6>" LOC = "AM18";
NET "PERIPHA<5>" LOC = "AL18";
NET "PERIPHA<4>" LOC = "AN21";
NET "PERIPHA<3>" LOC = "AM22";
NET "PERIPHA<2>" LOC = "AN19";
NET "PERIPHA<1>" LOC = "AM20";
NET "PDATA<31>" LOC = "C12";
NET "PDATA<30>" LOC = "E11";
NET "PDATA<29>" LOC = "D11";
NET "PDATA<28>" LOC = "C11";
NET "PDATA<27>" LOC = "B11";
NET "PDATA<26>" LOC = "A11";
NET "PDATA<25>" LOC = "E10";
NET "PDATA<24>" LOC = "C10";
NET "PDATA<23>" LOC = "B10";
NET "PDATA<22>" LOC = "D9";
NET "PDATA<21>" LOC = "C9";
NET "PDATA<20>" LOC = "A9";
NET "PDATA<19>" LOC = "E8";
NET "PDATA<18>" LOC = "D8";
NET "PDATA<17>" LOC = "C8";
NET "PDATA<16>" LOC = "B8";
NET "PDATA<15>" LOC = "A8";
NET "PDATA<14>" LOC = "D7";
NET "PDATA<13>" LOC = "C7";
NET "PDATA<12>" LOC = "B7";
NET "PDATA<11>" LOC = "C6";
NET "PDATA<10>" LOC = "A6";
NET "PDATA<9>" LOC = "E3";
NET "PDATA<8>" LOC = "B5";
NET "PDATA<7>" LOC = "A5";
NET "PDATA<6>" LOC = "B4";
NET "PDATA<5>" LOC = "A4";
NET "PDATA<4>" LOC = "C5";
NET "PDATA<3>" LOC = "D2";
NET "PDATA<2>" LOC = "A3";
NET "PDATA<1>" LOC = "D3";
```

```
NET "PDATA<0>" LOC = "E2";
NET "PCITEST" LOC = "M5";
NET "PCISTROBE" LOC = "D15";
NET "PCISELECT" LOC = "N4";
NET "PCISDA" LOC = "P4";
NET "PCISCL" LOC = "N3";
NET "PCIRSTOUT_L" LOC = "M4";
NET "PCIRSTOUTD" LOC = "L5";
NET "PCIRDY_OUT" LOC = "E13";
NET "PCIRDY_IN" LOC = "C14";
NET "PCIPCLKOUT" LOC = "D14";
NET "PCIIRQ_OUT" LOC = "J5";
NET "PCIIRQ_IN" LOC = "P2";
NET "PCICLKIN" LOC = "C15";
NET "PCIALE" LOC = "K4";
NET "PBLAST" LOC = "D12";
NET "PA_L" LOC = "C18";
NET "PAGE" LOC = "V30";
NET "PADDR<18>" LOC = "C1";
NET "PADDR<17>" LOC = "F1";
NET "PADDR<16>" LOC = "B16";
NET "PADDR<15>" LOC = "D16";
NET "PADDR<14>" LOC = "E15";
NET "PADDR<13>" LOC = "E16";
NET "PADDR<12>" LOC = "B17";
NET "PADDR<11>" LOC = "C17";
NET "PADDR<10>" LOC = "J4";
NET "PADDR<9>" LOC = "H4";
NET "PADDR<8>" LOC = "G4";
NET "PADDR<7>" LOC = "F4";
NET "PADDR<6>" LOC = "H5";
NET "PADDR<5>" LOC = "G5";
NET "PADDR<4>" LOC = "F5";
NET "PADDR<3>" LOC = "A15";
NET "PADDR<2>" LOC = "E14";
NET "PADDR<1>" LOC = "C16";
NET "PADDR<0>" LOC = "H3";
NET "MPD<6>" LOC = "W33";
NET "MPD<5>" LOC = "Y32";
NET "MPD<4>" LOC = "AA31";
NET "MPD<3>" LOC = "Y30";
NET "MPD<2>" LOC = "AA29";
NET "MPD<1>" LOC = "Y29";
NET "MPCLK" LOC = "AA33";
NET "JPERIPHD<48>" LOC = "AK16";
NET "JPERIPHD<47>" LOC = "AJ16";
NET "JPERIPHD<46>" LOC = "AK15";
NET "JPERIPHD<45>" LOC = "AJ15";
NET "JPERIPHD<44>" LOC = "AK14";
NET "JPERIPHD<43>" LOC = "AJ14";
NET "JPERIPHD<42>" LOC = "AK12";
NET "JPERIPHD<41>" LOC = "AJ12";
NET "JPERIPHD<40>" LOC = "AJ11";
NET "JPERIPHD<39>" LOC = "AK10";
NET "JPERIPHD<38>" LOC = "AJ10";
NET "JPERIPHD<37>" LOC = "AK9";
NET "JPERIPHD<36>" LOC = "AJ9";
NET "JPERIPHD<35>" LOC = "AJ8";
NET "JPERIPHD<34>" LOC = "AK7";
NET "JPERIPHD<33>" LOC = "AJ7";
NET "JPERIPHD<32>" LOC = "AK6";
NET "JPERIPHD<31>" LOC = "AJ6";
NET "JPERIPHD<30>" LOC = "AK5";
NET "JPERIPHD<29>" LOC = "AJ2";
NET "JPERIPHD<28>" LOC = "AJ3";
NET "JPERIPHD<27>" LOC = "AH3";
NET "JPERIPHD<26>" LOC = "AH1";
NET "JPERIPHD<25>" LOC = "AG5";
NET "JPERIPHD<24>" LOC = "AG4";
NET "JPERIPHD<23>" LOC = "AG2";
NET "JPERIPHD<22>" LOC = "AF5";
NET "JPERIPHD<21>" LOC = "AF3";
NET "JPERIPHD<20>" LOC = "AF4";
NET "JPERIPHD<19>" LOC = "AF2";
NET "JPERIPHD<18>" LOC = "AE5";
NET "JPERIPHD<17>" LOC = "AE3";
NET "JPERIPHD<16>" LOC = "AE4";
NET "JPERIPHD<15>" LOC = "AE1";
NET "JPERIPHD<14>" LOC = "AD5";
NET "JPERIPHD<13>" LOC = "AD4";
NET "JPERIPHD<12>" LOC = "AD3";
NET "JPERIPHD<11>" LOC = "AC5";
NET "JPERIPHD<10>" LOC = "AC3";
NET "JPERIPHD<9>" LOC = "AC1";
```

```
NET "JPERIPHD<8>" LOC = "AB4";
NET "JPERIPHD<7>" LOC = "AB3";
NET "JPERIPHD<6>" LOC = "AA5";
NET "JPERIPHD<5>" LOC = "AA3";
NET "JPERIPHD<4>" LOC = "AA1";
NET "JPERIPHD<3>" LOC = "Y5";
NET "JPERIPHD<2>" LOC = "Y4";
NET "JPERIPHD<1>" LOC = "Y3";
NET "IRQ2_L" LOC = "E28";
NET "IRQ1_L" LOC = "E20";
NET "IRQ0_L" LOC = "E27";
NET "ID<2>" LOC = "U33";
NET "ID<1>" LOC = "U32";
NET "ID<0>" LOC = "R33";
NET "HSCLK" LOC = "AL17";
NET "HEXLATCH" LOC = "V4";
NET "HEXBLANK" LOC = "W5";
NET "HBR_L" LOC = "P31";
NET "HBG_L" LOC = "T31";
NET "FO2VLTN" LOC = "AN26";
NET "FO2TXSCD" LOC = "AM31";
NET "FO2TXHALT" LOC = "AM27";
NET "FO2TXFULL" LOC = "AL19";
NET "FO2TXEMPTY" LOC = "AM29";
NET "FO2TXDATA<7>" LOC = "AK24";
NET "FO2TXDATA<6>" LOC = "AJ24";
NET "FO2TXDATA<5>" LOC = "AL23";
NET "FO2TXDATA<4>" LOC = "AK23";
NET "FO2TXDATA<3>" LOC = "AJ23";
NET "FO2TXDATA<2>" LOC = "AK22";
NET "FO2TXDATA<1>" LOC = "AJ22";
NET "FO2TXDATA<0>" LOC = "AL21";
NET "FO2TXCMD<3>" LOC = "AL25";
NET "FO2TXCMD<2>" LOC = "AL30";
NET "FO2TXCMD<1>" LOC = "AL28";
NET "FO2TXCMD<0>" LOC = "AL29";
NET "FO2RXSCD" LOC = "AM23";
NET "FO2RXEMPTY" LOC = "AK21";
NET "FO2RXDATA<7>" LOC = "AL24";
NET "FO2RXDATA<6>" LOC = "AJ25";
NET "FO2RXDATA<5>" LOC = "AK25";
NET "FO2RXDATA<4>" LOC = "AJ26";
NET "FO2RXDATA<3>" LOC = "AK26";
NET "FO2RXDATA<2>" LOC = "AK27";
NET "FO2RXDATA<1>" LOC = "AL26";
NET "FO2RXDATA<0>" LOC = "AM30";
NET "FO2RXCMD<3>" LOC = "AJ19";
NET "FO2RXCMD<2>" LOC = "AL20";
NET "FO2RXCMD<1>" LOC = "AJ21";
NET "FO2RXCMD<0>" LOC = "AJ20";
NET "FO2RXCLK" LOC = "A17";
NET "FO2RESET" LOC = "AJ27";
NET "FO2LFI" LOC = "AM24";
NET "FO1VLTN" LOC = "AK32";
NET "FO1TXSCD" LOC = "AL33";
NET "FO1TXHALT" LOC = "AK31";
NET "FO1TXEMPTY" LOC = "AJ31";
NET "FO1TXDATA<7>" LOC = "AE30";
NET "FO1TXDATA<6>" LOC = "AD31";
NET "FO1TXDATA<5>" LOC = "AE29";
NET "FO1TXDATA<4>" LOC = "AD30";
NET "FO1TXDATA<3>" LOC = "AC33";
NET "FO1TXDATA<2>" LOC = "AD29";
NET "FO1TXDATA<1>" LOC = "AC31";
NET "FO1TXDATA<0>" LOC = "AC30";
NET "FO1TXCMD<3>" LOC = "AH29";
NET "FO1TXCMD<2>" LOC = "AH31";
NET "FO1TXCMD<1>" LOC = "AH32";
NET "FO1TXCMD<0>" LOC = "AG33";
NET "FO1RXSCD" LOC = "AH33";
NET "FO1RXEMPTY" LOC = "AN29";
NET "FO1RXDATA<7>" LOC = "AE33";
NET "FO1RXDATA<6>" LOC = "AE32";
NET "FO1RXDATA<5>" LOC = "AF30";
NET "FO1RXDATA<4>" LOC = "AF29";
NET "FO1RXDATA<3>" LOC = "AG30";
NET "FO1RXDATA<2>" LOC = "AG29";
NET "FO1RXDATA<1>" LOC = "AF31";
NET "FO1RXDATA<0>" LOC = "AF32";
NET "FO1RXCMD<3>" LOC = "AB29";
NET "FO1RXCMD<2>" LOC = "AB31";
NET "FO1RXCMD<1>" LOC = "AC29";
NET "FO1RXCMD<0>" LOC = "AB30";
NET "FO1RXCLK" LOC = "D17";
```

```
NET "FO1RESET" LOC = "AJ30";
NET "FO1LFI" LOC = "AN31";
NET "FORCLK" LOC = "AJ18";
NET "FLAG<3>" LOC = "D26";
NET "FLAG<2>" LOC = "E26";
NET "FLAG<1>" LOC = "E23";
NET "FLAG<0>" LOC = "E22";
NET "EBOOT" LOC = "V31";
NET "DT1" LOC = "E19";
NET "DSPTIMEXP_L" LOC = "D20";
NET "DSPRESET_L" LOC = "K31";
NET "DSPCLKEN_L" LOC = "E17";
NET "DSPCLK" LOC = "AJ17";
NET "DR1" LOC = "E25";
NET "DMAR1_L" LOC = "V32";
NET "DMAG1_L" LOC = "V29";
NET "DIPTRST" LOC = "U1";
NET "DFLASHWE_L" LOC = "W29";
NET "DFLASHOE_L" LOC = "W30";
NET "DFLASHCE_L" LOC = "W31";
NET "DATA<31>" LOC = "D28";
NET "DATA<30>" LOC = "B30";
NET "DATA<29>" LOC = "A31";
NET "DATA<28>" LOC = "A28";
NET "DATA<27>" LOC = "C28";
NET "DATA<26>" LOC = "C30";
NET "DATA<25>" LOC = "B29";
NET "DATA<24>" LOC = "B26";
NET "DATA<23>" LOC = "C26";
NET "DATA<22>" LOC = "A25";
NET "DATA<21>" LOC = "C27";
NET "DATA<20>" LOC = "D23";
NET "DATA<19>" LOC = "D21";
NET "DATA<18>" LOC = "C20";
NET "DATA<17>" LOC = "A27";
NET "DATA<16>" LOC = "B25";
NET "DATA<15>" LOC = "B20";
NET "DATA<14>" LOC = "A23";
NET "DATA<13>" LOC = "B24";
NET "DATA<12>" LOC = "D25";
NET "DATA<11>" LOC = "D24";
NET "DATA<10>" LOC = "C19";
NET "DATA<9>" LOC = "C23";
NET "DATA<8>" LOC = "D19";
NET "DATA<7>" LOC = "C29";
NET "DATA<6>" LOC = "C21";
NET "DATA<5>" LOC = "D27";
NET "DATA<4>" LOC = "B22";
NET "DATA<3>" LOC = "D22";
NET "DATA<2>" LOC = "B21";
NET "DATA<1>" LOC = "D30";
NET "DATA<0>" LOC = "C25";
NET "DACRST" LOC = "V3";
NET "DACMSB" LOC = "U2";
NET "DACLDB" LOC = "W1";
NET "DACLDA" LOC = "W3";
NET "DACCS" LOC = "V2";
NET "DACAB" LOC = "U4";
NET "CS_L" LOC = "P30";
NET "BUSMODE_L<4>" LOC = "T4";
NET "BUSMODE_L<4>" IOSTANDARD = PCI33_5;
NET "BUSMODE_L<3>" LOC = "T5";
NET "BUSMODE_L<3>" IOSTANDARD = PCI33_5;
NET "BUSMODE_L<2>" LOC = "U3";
NET "BUSMODE_L<2>" IOSTANDARD = PCI33_5;
NET "BUSMODE_L<1>" LOC = "T3";
NET "BUSMODE_L<1>" IOSTANDARD = PCI33_5;
NET "BRST" LOC = "R30";
NET "BMS_L" LOC = "T32";
NET "ADDR<31>" LOC = "P29";
NET "ADDR<30>" LOC = "N31";
NET "ADDR<29>" LOC = "K30";
NET "ADDR<28>" LOC = "G29";
NET "ADDR<27>" LOC = "N30";
NET "ADDR<26>" LOC = "K29";
NET "ADDR<25>" LOC = "G30";
NET "ADDR<24>" LOC = "M32";
NET "ADDR<23>" LOC = "H29";
NET "ADDR<22>" LOC = "F29";
NET "ADDR<21>" LOC = "M31";
NET "ADDR<20>" LOC = "J31";
NET "ADDR<19>" LOC = "M30";
NET "ADDR<18>" LOC = "F30";
NET "ADDR<17>" LOC = "J30";
```

```
NET "ADDR<16>" LOC = "M29";
NET "ADDR<15>" LOC = "J29";
NET "ADDR<14>" LOC = "F31";
NET "ADDR<13>" LOC = "L30";
NET "ADDR<12>" LOC = "H31";
NET "ADDR<11>" LOC = "L31";
NET "ADDR<10>" LOC = "F33";
NET "ADDR<9>" LOC = "H33";
NET "ADDR<8>" LOC = "L32";
NET "ADDR<7>" LOC = "E30";
NET "ADDR<6>" LOC = "E32";
NET "ADDR<5>" LOC = "J33";
NET "ADDR<4>" LOC = "E33";
NET "ADDR<3>" LOC = "C33";
NET "ADDR<2>" LOC = "D32";
NET "ADDR<1>" LOC = "G32";
NET "ADDR<0>" LOC = "L29";
NET "ACK" LOC = "R31";
CONFIG PROHIBIT = "D4";
CONFIG PROHIBIT = "E4";
CONFIG PROHIBIT = "K3";
CONFIG PROHIBIT = "L4";
CONFIG PROHIBIT = "P3";
CONFIG PROHIBIT = "W4";
CONFIG PROHIBIT = "AB5";
CONFIG PROHIBIT = "AC4";
CONFIG PROHIBIT = "AJ4";
CONFIG PROHIBIT = "D6";
CONFIG PROHIBIT = "A2";
NET "FO2RXCLK" TNM_NET = "FO2RXCLK";
TIMESPEC "TS_FO2RXCLK" = PERIOD "FO2RXCLK" 25 MHZ HIGH
50 %;
NET "ULCTL<1>" PULLUP;
NET "ULCTL<2>" PULLUP;
NET "ULCTL<3>" PULLUP;
NET "ULCTL<4>" PULLUP;
NET "ULCTL<5>" PULLUP;
NET "ULCTL<6>" PULLUP;
NET "ULCTL<7>" PULLUP;
NET "ULCTL<8>" PULLUP;
NET "ULCTL<9>" PULLUP;
NET "ULCTL<10>" PULLUP;
NET "MPCLK" PULLUP;
NET "MPD<1>" PULLUP;
NET "MPD<2>" PULLUP;
NET "MPD<3>" PULLUP;
NET "MPD<4>" PULLUP;
NET "MPD<5>" PULLUP;
NET "MPD<6>" PULLUP;
NET "PDATA<0>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<1>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<2>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<3>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<4>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<5>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<6>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<7>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<8>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<9>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<10>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<11>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<12>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<13>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<14>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<15>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<16>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<17>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<18>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<19>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<20>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<21>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<22>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<23>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<24>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<25>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<26>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<27>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<28>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<29>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<30>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "PDATA<31>" OFFSET = IN 20 ns BEFORE "HSCLK";
NET "DATA<0>" PULLUP;
NET "DATA<1>" PULLUP;
NET "DATA<2>" PULLUP;
```

```
NET "DATA<3>" PULLUP;
NET "DATA<4>" PULLUP;
NET "DATA<5>" PULLUP;
NET "DATA<6>" PULLUP;
NET "DATA<7>" PULLUP;
NET "DATA<8>" PULLUP;
NET "DATA<9>" PULLUP;
NET "DATA<10>" PULLUP;
NET "DATA<11>" PULLUP;
NET "DATA<12>" PULLUP;
NET "DATA<13>" PULLUP;
NET "DATA<14>" PULLUP;
NET "DATA<15>" PULLUP;
NET "DATA<16>" PULLUP;
NET "DATA<17>" PULLUP;
NET "DATA<18>" PULLUP;
NET "DATA<19>" PULLUP;
NET "DATA<20>" PULLUP;
NET "DATA<21>" PULLUP;
NET "DATA<22>" PULLUP;
NET "DATA<23>" PULLUP;
NET "DATA<24>" PULLUP;
NET "DATA<25>" PULLUP;
NET "DATA<26>" PULLUP;
NET "DATA<27>" PULLUP;
NET "DATA<28>" PULLUP;
NET "DATA<29>" PULLUP;
NET "DATA<30>" PULLUP;
NET "DATA<31>" PULLUP;
NET "ADDR<0>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<1>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<2>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<3>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<4>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<5>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<6>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<7>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<8>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<9>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<10>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<11>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<12>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<13>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<14>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<15>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<16>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<17>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<18>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<19>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<20>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<21>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<22>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<23>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<24>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<25>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<26>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<27>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<28>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<29>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<30>" OFFSET = IN 10 ns BEFORE "HSCLK";
NET "ADDR<31>" OFFSET = IN 10 ns BEFORE "HSCLK";
#NET "PCICLKIN" PULLUP;
```

9.2.3 CommStd.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package COMMSTD is

  subtype T_COMMAND is std_logic_vector( 3 downto 0 );
  subtype T_EX_CMD is std_logic_vector( 7 downto 0 );

  subtype T_BYTE is std_logic_vector( 7 downto 0 );
  subtype T_DATA is std_logic_vector( 15 downto 0 );

  type T_DATA_PAYLOAD is array ( 9 downto 0 ) of T_BYTE;
  type T_PACKETBUFFER is array ( 0 to 19 ) of T_BYTE;

  subtype T_NETTIME is std_logic_vector( 7 downto 0 );
  subtype T_NODEADDR is std_logic_vector( 7 downto 0 );

  --The following refers to the scheduled data
  constant C_ACTIVE_DATA_MAX: natural := 15;
  type T_SYNC_ATTR_ARRAY is array (C_ACTIVE_DATA_MAX
downto 0 ) of T_DATA;

  --The following refers to the normal data
  constant C_NORMAL_DATA_MAX: natural := 3;
  type T_NORMAL_DATA_ARRAY is array (C_NORMAL_DATA_MAX
downto 0) of T_DATA;

  --The following refers to the unscheduled data
  constant C_ASYNC_DATA_MAX: natural := 15;
  type T_ASYNC_ATTR_ARRAY is array (C_ASYNC_DATA_MAX
downto 0) of T_DATA;

  --The following refers to the DPRAM
  constant DPRAM_DATA_WIDTH: natural    := 16;
  constant DPRAM_ADDR_WIDTH: natural := 5;
  constant DPRAM_MEM_DEPTH: natural := 32;

  --The following refers to the mode that PESNet is in
  type E_PACKET_MODE is ( SHORT_PKT, LONG_PKT);

  --The following are the COMMANDS for PESNet
  constant C_CMD_SETVAL_SYNC      : T_COMMAND := "0100";
--sync dm
  constant C_CMD_GETVAL_SYNC      : T_COMMAND := "0101";
--sync dm
  constant C_CMD_NULL             : T_COMMAND := "0110";
  constant C_CMD_EVENT     : T_COMMAND := "1000";  --
event mgr
  constant C_CMD_NORMAL    : T_COMMAND := "0001";  --
normal dm
  constant C_CMD_GETVAL_ASYNC     : T_COMMAND := "0010";
--asynchronous dm
  constant C_CMD_SETVAL_ASYNC     : T_COMMAND := "0011";
--asynchronous dm
  constant C_CMD_EXTENDED         : T_COMMAND := "0111";
--extended dm


end COMMSTD;

package body COMMSTD is
end COMMSTD;
```

9.2.4 CommStack.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity commstack is
    Port (
    --Global inputs
    BLOCK_HSCLK: in std_logic;
    BLOCK_RESET_L: in std_logic;

    --Event manager outputs
    FAULT_INJ: out std_logic;                    -- fault
injection for fault protection testing
    FAULT_SOURCE_L: in std_logic_vector (4 downto 0); --
current fault sources
    FAULT_RESET_L: out std_logic_vector (4 downto 0); --
resets appropriate faults

    --Synchronous data manager outputs
    ACTIVE_DATA_SYNC: out T_SYNC_ATTR_ARRAY;

    --Normal data manager signals
    STATUS_DATA: in T_NORMAL_DATA_ARRAY;
    ACTIVE_DATA_NORMAL: out T_NORMAL_DATA_ARRAY;

    PKT_ALIGNMENT: in std_logic_vector( 4 downto 0 );
    OPEN_LOOP_MODE_INDEX: in std_logic;
    SYS_RST_COMM: out std_logic;
    --packet coming in from redirector
    VLTN: in std_logic;
    LFI_L: in std_logic;
    TXDATA1: out std_logic_vector( 7 downto 0);
    TXCMD1: out std_logic_vector( 3 downto 0 );
    TXSCD1: out std_logic;
    REFCLK1: in std_logic;
    TXFULL_L: in std_logic;
    RXDATA1: in std_logic_vector( 7 downto 0);
    RXCMD1: in std_logic_vector( 3 downto 0 );
    RXSCD1: in std_logic;
    RXCLK1: in std_logic;
    -- Environmental info
    MAS_ADDR:       in T_NODEADDR;

    PKT_MODE: in E_PACKET_MODE;

    --interface to DSP
    BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
    BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0 );
    BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );
    BLOCK_WRITE_L: in std_logic;
    BLOCK_ENABLE: in std_logic;
    BLOCK_DONE: out std_logic;
    NET_TIME_OUT: out std_logic_vector( 7 downto 0 );
    SYNC: out std_logic;

    ASYNC_DATA_OUT: out T_ASYNC_ATTR_ARRAY;
    ASYNC_DATA_IN: in T_ASYNC_ATTR_ARRAY;
    ASYNC_DATA_DIR: in std_logic_vector(
C_ASYNC_DATA_MAX downto 0 );  -- 1 = data comes from
packet, 0 = device

    PKT_LENGTH: in std_logic_vector( 4 downto 0 );
    PIN_RST: in std_logic;

    DEBUG_DEF: out std_logic_vector( 7 downto 0 );
    DEBUG_FRM: out std_logic_vector( 7 downto 0 );

    DEBUG_AUX: out std_logic_vector( 32 downto 1 )

   );
end commstack;
```

```
architecture Behavioral of commstack is

--=======================================================
component asynchronous_dm is
    Port (
   BLOCK_HSCLK: in std_logic;  --clock input
   BLOCK_RESET_L: in std_logic;  --reset signal

   BLOCK_PKT_GOOD: in std_logic;

   BLOCK_PKT_RDY: out std_logic;
   BLOCK_PKT_CLR: in std_logic;

   NET_TIME: in T_NETTIME;

   BLOCK_PKT_CMD_IN: in T_COMMAND;
   BLOCK_PKT_FROM_ADDR_IN: in T_NODEADDR;
   BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;

   BLOCK_PKT_CMD_OUT: out T_COMMAND;
   BLOCK_PKT_TO_ADDR_OUT: out T_NODEADDR;
   BLOCK_PKT_DATA_OUT: out T_DATA_PAYLOAD;

   ASYNC_DATA_OUT: out T_ASYNC_ATTR_ARRAY;
   ASYNC_DATA_IN: in T_ASYNC_ATTR_ARRAY;

   ASYNC_DATA_DIR: in std_logic_vector( C_ASYNC_DATA_MAX
downto 0 )  -- 1 = data comes from packet, 0 = device
   );
end component;

--=======================================================
component cmdproc is
    Port (
   BLOCK_RESET_L: in std_logic;
   BLOCK_HSCLK: in std_logic;

   --packet sent to redirector
   REDIR_DOWN_DATA:       out T_DATA_PAYLOAD;
   REDIR_DOWN_ADDR_TO:    out T_NODEADDR;
   REDIR_DOWN_ADDR_FROM:  out T_NODEADDR;
   REDIR_DOWN_CMD:     out T_COMMAND;

   --packet coming in from redirector
   REDIR_UP_CMD:       in T_COMMAND;
   REDIR_UP_ADDR_TO:   in T_NODEADDR;
   REDIR_UP_ADDR_FROM: in T_NODEADDR;
   REDIR_UP_DATA:   in T_DATA_PAYLOAD;

   -- Environmental info
   THIS_ADDR:       in T_NODEADDR;
   MAS_ADDR:        in T_NODEADDR;

   -- Info on packet status
   BLOCK_PKT_GOOD: in std_logic;
   MGR_PKT_GOOD: out std_logic;

   -- Event Mgr
   EMGR_PKT_RDY:    in std_logic;
   EMGR_PKT_CLR:    out std_logic;
   EMGR_CMD_OUT:    in T_COMMAND;
   EMGR_DATA_OUT: in T_DATA_PAYLOAD;

   -- Scheduled Data Mgr
   SCHED_PKT_RDY:   in std_logic;
   SCHED_PKT_CLR:   out std_logic;
   SCHED_CMD_OUT:   in T_COMMAND;
   SCHED_DATA_OUT:  in T_DATA_PAYLOAD;
   SCHED_ADDR_TO_OUT:  in T_NODEADDR;

   -- Normal Data Mgr
   NORMAL_PKT_RDY:  in std_logic;
   NORMAL_PKT_CLR:  out std_logic;
   NORMAL_CMD_OUT:  in T_COMMAND;
   NORMAL_DATA_OUT:    in T_DATA_PAYLOAD;
   NORMAL_ADDR_TO_OUT: in T_NODEADDR;


   -- Unscheduled Data Mgr
   UNSCHED_PKT_RDY:    in std_logic;
   UNSCHED_PKT_CLR:    out std_logic;
   UNSCHED_CMD_OUT:    in T_COMMAND;
   UNSCHED_DATA_OUT: in T_DATA_PAYLOAD;
   UNSCHED_ADDR_TO_OUT: in T_NODEADDR;
```

```
    -- DDM
    DDM_PKT_RDY: in std_logic;
    DDM_PKT_CLR: out std_logic;
    DDM_CMD_OUT: in T_COMMAND;
    DDM_ADDR_OUT: in T_NODEADDR;
    DDM_DATA_OUT: in T_DATA_PAYLOAD;
    DDM_ADDR_FROM_OUT: in T_NODEADDR;

    --Extended data manage
    EXT_PKT_RDY: in std_logic;
    EXT_PKT_CLR: out std_logic;
    EXT_CMD_OUT: in T_COMMAND;
    EXT_ADDR_OUT: in T_NODEADDR;
    EXT_DATA_OUT: in T_DATA_PAYLOAD;
    EXT_ADDR_FROM_OUT: in T_NODEADDR;
    EXT_PKT_ABSORB: in std_logic;

    COPY_PKT: in std_logic;
    PKT_RDY: out std_logic;

    DEBUG_CMDPROC: out std_logic_vector( 7 downto 0 )
   );
end component;
--=========================================================
component deframer is
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;

    RXDATA: in std_logic_vector( 7 downto 0 );
    RXCMD: in std_logic_vector( 3 downto 0 );
    RXSCD: in std_logic;

    RXCLK: in std_logic;

    VLTN: in std_logic;
    LFI_L: in std_logic;


    STARTFRAME: out std_logic;
```

```
    BLOCK_PKT_GOOD: out std_logic;
         --From bottom layer
    REDIR_INX_CMD: out T_COMMAND;
    REDIR_INX_NETTIME: out T_NETTIME;
    REDIR_INX_ADDR_TO:  out T_NODEADDR;
    REDIR_INX_ADDR_FROM: out T_NODEADDR;
    REDIR_INX_DATA: out T_DATA_PAYLOAD;
    POSCTR_OUT: out std_logic_vector( 4 downto 0 );
    DEBUG_DEF: out std_logic_vector( 7 downto 0 )

    );
end component;
--=========================================================
component DDM2 is
  port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_WRITE_L: in std_logic;
  BLOCK_ENABLE: in std_logic;
  BLOCK_DONE: out std_logic;

  THIS_ADDR: in T_NODEADDR;
  NETTIME: in T_NETTIME;

  --interface from command processor
  BLOCK_PKT_RDY: out std_logic;
  BLOCK_PKT_CLR: in std_logic;

  BLOCK_PKT_ADDR_TO_OUT: out T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_OUT: out T_NODEADDR;
  BLOCK_PKT_CMD_OUT: out T_COMMAND;
  BLOCK_PKT_DATA_OUT: out T_DATA_PAYLOAD;

  BLOCK_PKT_ADDR_TO_IN: in T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_IN: in T_NODEADDR;
  BLOCK_PKT_CMD_IN: in T_COMMAND;
  BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;
     BLOCK_PKT_GOOD: in std_logic;

  --interface to DSP
  BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
```

```
  BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0 );
  BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );

  DEBUG_DDM: out std_logic_vector( 15 downto 0 )
);

end component;


--=======================================================
component event_man is
    Port (
     BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;    -- goes high
when packet is complete
    BLOCK_PKT_GOOD: in std_logic;               --signals
packet is for this block
    BLOCK_PKT_RDY: out std_logic;               -- goes
high when an output packet is ready
    BLOCK_PKT_CLR: in std_logic;         -- clears ready
flag
    FREEZE: in std_logic;                       -- stops
output from changing, while command processor is
clearing ready flag

    BLOCK_CMD_IN: in T_COMMAND;   -- input command
(should always = BLOCK_CMD_IN if packet is for this
block)
    BLOCK_CMD_OUT: out T_COMMAND; -- always =
BLOCK_CMD_IN if for this block
    NET_TIME: in T_NETTIME;              -- current time
    BLOCK_DATA_IN: in T_DATA_PAYLOAD;    -- input
package, BLOCK_DATA_IN(1) = x'01' for fault injection
                                 -- = x'00' for fault
reset and BLOCK_DATA_IN(2) is where the reset vector is
located
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;  --
BLOCK_DATA_OUT(2) is the current fault source vector,
                                 --
BLOCK_DATA_OUT(3) is the current time
    -- fault injection for fault protection testing
    FAULT_INJ: out std_logic;                   --
injects a fault into the fault block
    FAULT_SOURCE_L: in std_logic_vector (4 downto 0); --
current fault sources (from fault block)
    FAULT_RESET_L: out std_logic_vector (4 downto 0);  --
faults to be reset
    PKT_MODE: in E_PACKET_MODE           -- long packet
or short packet
    );
end component;
--=======================================================
component EXTENDED_MGR is
port (
  BLOCK_HSCLK:          in std_logic;
  BLOCK_RESET_L:        in std_logic;

  NET_TIME_OVERRIDE:       out T_NETTIME;
  NET_TIME_SET:         out std_logic;

  BLOCK_PKT_GOOD:       in std_logic;
  BLOCK_PKT_RDY:        out std_logic;
  BLOCK_PKT_CLR:        in std_logic;

  BLOCK_PKT_ADDR_TO_OUT:   out T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_OUT: out T_NODEADDR;
  BLOCK_PKT_CMD_OUT:       out T_COMMAND;
  BLOCK_PKT_DATA_OUT:   out T_DATA_PAYLOAD;

  BLOCK_PKT_ADDR_TO_IN:    in T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_IN:  in T_NODEADDR;
  BLOCK_PKT_CMD_IN:     in T_COMMAND;
  BLOCK_PKT_DATA_IN:       in T_DATA_PAYLOAD;
  SYS_RST_COMM: out std_logic;

  LOAD_CLK_VAL:         out std_logic;
  NEW_CLK_VAL:          out T_NETTIME;
  PIN_RST: in std_logic;

  BLOCK_PKT_ABSORB:     out std_logic;

  THIS_ADDR:         out T_NODEADDR       );
```

```
end component;

--=======================================================
component NETCLK_MGR is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  SETCLK: in std_logic;
  SETVAL: in T_NETTIME;

  INCREMENT: in std_logic;

  CURRTIME: out T_NETTIME;
  NEXTTIME: out T_NETTIME
);
end component;

--=======================================================
component fault
    Port (
    BLOCK_RESET_L: in std_logic;    -- signal that resets
teh fault protection
    BLOCK_HSCLK: in std_logic;     -- High speed clock (80
MHz)
    ENABLE: in std_logic; -- active high enable
    FAULT_RESET_L: in std_logic_vector (4 downto 0);  --
resets fault protection
    PWM_IN: in std_logic_vector (1 downto 0);-- PWM
switch inputs
    FAULT_IPM: in std_logic;       -- fault signal from
IPM module
    FAULT_COMM: in std_logic;       -- communication fault
input
    FAULT_INJ: in std_logic; -- fault injection
    TEMP: in std_logic_vector (12 downto 0); --
Temperature from A/D converter
    TEMP_REF: in std_logic_vector (12 downto 0);      --
reference temp (if temp > temp_ref cause a fault)
    -- PWM deadtime reference (if deadtime > deadtime
ref, cause a fault)
    DEADTIME_REF: in std_logic_vector (9 downto 0);
    -- outputs
    PWM_OUT: out std_logic_vector (1 downto 0); -- PWM
switching outputs
    PWM_ENABLE_OUT: out std_logic;       -- PWM enabled
    -- fault source (11111=no fault, 11110= IPM fault,
11101=comm fault
    -- 11011= fault injection, 10111= temp fault, 01111=
deadtime fault)
    FAULT_SOURCE_L_OUT: out std_logic_vector (4 downto 0)
  );
end component;

--=======================================================
component framer is
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;

    --Interface to CY7C9689A-AC transmit side
    TXDATA: out std_logic_vector( 7 downto 0 );
    TXCMD: out std_logic_vector( 3 downto 0 );
    TXSCD: out std_logic;
    REFCLK: in std_logic;
    TXFULL_L: in std_logic;

    --Indication to start sending new frame
    STARTFRAME: in std_logic;

    --Packet data to use when sending frame
    REDIR_OUTX_CMD: in T_COMMAND;
    REDIR_OUTX_ADDR_TO:  in T_NODEADDR;
    REDIR_OUTX_ADDR_FROM: in T_NODEADDR;
    REDIR_OUTX_DATA: in T_DATA_PAYLOAD;
    NETTIME: in T_NETTIME;
    POSCTR_OUT: out std_logic_vector( 4 downto 0 );

    COPY_PKT: out std_logic;
    PKT_RDY: in std_logic;

    DEBUG_FRM: out std_logic_vector( 7 downto 0 )

    );
```

```
end component;
--=======================================================
component indexmgr is
  port(BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_PKT_GOOD: in std_logic;
  START_FRAME: out std_logic;
  IN_POSCTR: in std_logic_vector( 4 downto 0 );
  OUT_POSCTR: in std_logic_vector( 4 downto 0 );

  PKT_LENGTH: in std_logic_vector( 4 downto 0 );
  OPEN_LOOP_MODE: in std_logic;
  PKT_ALIGNMENT: in std_logic_vector( 4 downto 0 );

  TXCLK: in std_logic;
  RXCLK: in std_logic);
end component;
--=======================================================

component SCHED_DM is
    Port (
    BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;
    BLOCK_PKT_GOOD: in std_logic;

    BLOCK_PKT_RDY: out std_logic;
    BLOCK_PKT_CLR: in std_logic;

    BLOCK_CMD: in T_COMMAND;
    NET_TIME: in T_NETTIME;

    BLOCK_CMD_OUT: out T_COMMAND;
    BLOCK_DATA_IN: in T_DATA_PAYLOAD;
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;
    BLOCK_FROM_ADDR_IN: in T_NODEADDR;
    BLOCK_TO_ADDR_OUT: out T_NODEADDR;

    ACTIVE_DATA: out T_SYNC_ATTR_ARRAY;
    PKT_MODE: in E_PACKET_MODE

   );
end component;

--=======================================================

component NORMAL_DM is
    Port (
    BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;
    BLOCK_PKT_GOOD: in std_logic;

    BLOCK_PKT_RDY: out std_logic;
    BLOCK_PKT_CLR: in std_logic;

    BLOCK_CMD_IN: in T_COMMAND;
    BLOCK_CMD_OUT: out T_COMMAND;
    NET_TIME: in T_NETTIME;

    BLOCK_DATA_IN: in T_DATA_PAYLOAD;
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;
    BLOCK_FROM_ADDR_IN: in T_NODEADDR;
    BLOCK_TO_ADDR_OUT: out T_NODEADDR;

    ACTIVE_DATA: out T_NORMAL_DATA_ARRAY;

    PKT_MODE: in E_PACKET_MODE;

    --Status/Sensor Information
    STATUS_DATA: in T_NORMAL_DATA_ARRAY;
    SYNC: out std_logic

   );
end component;
--=======================================================
component redirector is
  port(
     --from top layer
     REDIR_OUT_CMD: out T_COMMAND;
```

```
    REDIR_OUT_ADDR_TO:  out T_NODEADDR;
    REDIR_OUT_ADDR_FROM: out T_NODEADDR;
    REDIR_OUT_DATA: out T_DATA_PAYLOAD;

    REDIR_IN_CMD: in T_COMMAND;
    REDIR_IN_ADDR_TO:  in T_NODEADDR;
    REDIR_IN_ADDR_FROM: in T_NODEADDR;
    REDIR_IN_DATA: in T_DATA_PAYLOAD;

    --From bottom layer
    REDIR_OUT1_CMD: out T_COMMAND;
    REDIR_OUT1_ADDR_TO:  out T_NODEADDR;
    REDIR_OUT1_ADDR_FROM: out T_NODEADDR;
    REDIR_OUT1_DATA: out T_DATA_PAYLOAD;

    REDIR_IN1_CMD: in T_COMMAND;
    REDIR_IN1_ADDR_TO:  in T_NODEADDR;
    REDIR_IN1_ADDR_FROM: in T_NODEADDR;
    REDIR_IN1_DATA: in T_DATA_PAYLOAD);


end component;

--=======================================================
--Signals for interconnecting components


  signal THIS_ADDR:               T_NODEADDR;
--Global Signals
  signal BLOCK_PKT_COMPLETE:            std_logic;
  signal BLOCK_PKT_GOOD:                std_logic;

  signal MGR_PKT_GOOD:            std_logic;
  signal MGR_PKT_GOODA:           std_logic;  --Delayed
for distribution purposes (speed up clock)

-- signal NET_TIME:
  T_NETTIME;
-- signal PKT_MODE:
  E_PACKET_MODE;


-- signal THIS_ADDR:
  T_NODEADDR;
-- signal MAS_ADDR:                     T_NODEADDR;

--Command Processor Signals
  --packet sent to redirector
-- signal REDIR_OUT_ADDR_TO:     T_NODEADDR;
-- signal REDIR_OUT_ADDR_FROM:   T_NODEADDR;
-- signal REDIR_OUT_CMD:                T_COMMAND;

  --packet coming in from redirector
-- signal REDIR_IN_CMD:                 T_COMMAND;
-- signal REDIR_IN_ADDR_TO:             T_NODEADDR;
-- signal REDIR_IN_ADDR_FROM:    T_NODEADDR;
-- signal REDIR_IN_DATA:                T_DATA_PAYLOAD;

--DDM signals
  signal DDM_PKT_RDY:                   std_logic;
  signal DDM_PKT_CLR:                   std_logic;
  signal DDM_CMD_OUT:                   T_COMMAND;
  signal DDM_DATA_OUT:            T_DATA_PAYLOAD;
  signal DDM_ADDR_TO_OUT:               T_NODEADDR;
  signal DDM_ADDR_FROM_OUT:             T_NODEADDR;
  signal DEBUG_DDM:               std_logic_vector( 15
downto 0 );

--Event Manager Signals
  signal EMGR_PKT_RDY:            std_logic;
  signal EMGR_PKT_CLR:            std_logic;
  signal EMGR_CMD_OUT:            T_COMMAND;
  signal EMGR_DATA_OUT:           T_DATA_PAYLOAD;

--Extended Manager
  signal EXT_PKT_RDY:                   std_logic;
  signal EXT_PKT_CLR:                   std_logic;
  signal EXT_CMD_OUT:                   T_COMMAND;
  signal EXT_DATA_OUT:            T_DATA_PAYLOAD;
  signal EXT_ADDR_TO_OUT:               T_NODEADDR;
  signal EXT_ADDR_FROM_OUT:             T_NODEADDR;
  signal EXT_PKT_ABSORB:                std_logic;

  signal LOAD_CLK_VAL:            std_logic;
```

```
  signal NEW_CLK_VAL:                    T_NETTIME;

--Normal Data Manager Signals
  signal NORMAL_PKT_RDY:          std_logic;
  signal NORMAL_PKT_CLR:                 std_logic;
  signal NORMAL_CMD_OUT:                 T_COMMAND;
  signal NORMAL_DATA_OUT:         T_DATA_PAYLOAD;
  signal NORMAL_ADDR_TO_OUT:      T_NODEADDR;

--Scheduled Data Manager Signals
  signal SCHED_DATA_OUT:                 T_DATA_PAYLOAD;
  signal SCHED_PKT_RDY:           std_logic;
  signal SCHED_PKT_CLR:           std_logic;
  signal SCHED_CMD_OUT:           T_COMMAND;
  signal SCHED_ADDR_TO_OUT:              T_NODEADDR;

-- Unscheduled Data Mgr
  signal UNSCHED_PKT_RDY:         std_logic;
  signal UNSCHED_PKT_CLR:         std_logic;
  signal UNSCHED_CMD_OUT:         T_COMMAND;
  signal UNSCHED_DATA_OUT:               T_DATA_PAYLOAD;
  signal UNSCHED_ADDR_TO_OUT:            T_NODEADDR;

--Redirector interface signals
  signal REDIR_OUT1_CMD:          T_COMMAND;
  signal REDIR_OUT1_ADDR_TO:      T_NODEADDR;
  signal REDIR_OUT1_ADDR_FROM:    T_NODEADDR;
  signal REDIR_OUT1_DATA:         T_DATA_PAYLOAD;

  signal REDIR_DOWN_CMD:          T_COMMAND;
  signal REDIR_DOWN_ADDR_TO:      T_NODEADDR;
  signal REDIR_DOWN_ADDR_FROM:    T_NODEADDR;
  signal REDIR_DOWN_DATA:         T_DATA_PAYLOAD;

  signal REDIR_UP_CMD:     T_COMMAND;
  signal REDIR_UP_CMDA:           T_COMMAND;

  signal REDIR_UP_ADDR_TO:        T_NODEADDR;
  signal REDIR_UP_ADDR_FROM:      T_NODEADDR;
  signal REDIR_UP_DATA:    T_DATA_PAYLOAD;

  signal REDIR_IN1_CMD:           T_COMMAND;
  signal REDIR_IN1_ADDR_TO:       T_NODEADDR;
  signal REDIR_IN1_ADDR_FROM:     T_NODEADDR;
  signal REDIR_IN1_DATA:          T_DATA_PAYLOAD;
  signal NET_TIME:          T_NETTIME;

  signal STARTFRAME:              std_logic; --start
transmitting a new frame

  signal REFCLK1X:          std_logic;

  signal RXDATA1X:          std_logic_vector( 7 downto
0);
  signal RXCMD1X:           std_logic_vector( 3 downto 0
);
  signal RXSCD1X:           std_logic;
  signal RXCLK1X:           std_logic;
  signal TXCLKX:            std_logic;
  signal TXFULL_LX:         std_logic;

  signal COPY_PKT:          std_logic;
  signal PKT_RDY:           std_logic;

  signal POSCTR_FRM:        std_logic_vector( 4 downto 0
);
  signal POSCTR_DEF:        std_logic_vector( 4 downto 0
);
-- signal DEBUG_DEFX:             std_logic_vector( 7
downto 0 );
-- signal DEBUG_FRMX:             std_logic_vector( 7
downto 0 );
  signal DEBUG_CMDPROC:     std_logic_vector( 7 downto 0
);

begin
BLOCK_PKT_COMPLETE <= BLOCK_PKT_GOOD;

process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     MGR_PKT_GOODA <= '0';
     REDIR_UP_CMDA <= "0000";
```

```
    else
     --note there is a one cycle acceptible lag after mgr
pkt good goes low
     MGR_PKT_GOODA <= MGR_PKT_GOOD;
     REDIR_UP_CMDA <= REDIR_UP_CMD;
    end if;
  end if;
end process;

--=======================================================
CASYNC_DM: ASYNCHRONOUS_DM port map(
  BLOCK_HSCLK           => BLOCK_HSCLK,
  BLOCK_RESET_L         => BLOCK_RESET_L,

  BLOCK_PKT_GOOD        => MGR_PKT_GOOD,
  BLOCK_PKT_RDY         => UNSCHED_PKT_RDY,
  BLOCK_PKT_CLR         => UNSCHED_PKT_CLR,

  BLOCK_PKT_CMD_IN      => REDIR_UP_CMD,
  BLOCK_PKT_FROM_ADDR_IN   => REDIR_UP_ADDR_FROM,
  BLOCK_PKT_DATA_IN     => REDIR_UP_DATA,

  BLOCK_PKT_CMD_OUT     => UNSCHED_CMD_OUT,
  BLOCK_PKT_TO_ADDR_OUT => UNSCHED_ADDR_TO_OUT,
  BLOCK_PKT_DATA_OUT    => UNSCHED_DATA_OUT,

  ASYNC_DATA_OUT        => ASYNC_DATA_OUT,
  ASYNC_DATA_IN         => ASYNC_DATA_IN,
  ASYNC_DATA_DIR        => ASYNC_DATA_DIR,

  NET_TIME          => NET_TIME
);

--=======================================================
CCMDPROC: cmdproc port map (
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L         => BLOCK_RESET_L,

  MGR_PKT_GOOD          => MGR_PKT_GOOD,
  BLOCK_PKT_GOOD        => BLOCK_PKT_GOOD,

  REDIR_UP_DATA         => REDIR_UP_DATA,
  REDIR_UP_ADDR_TO      => REDIR_UP_ADDR_TO,
  REDIR_UP_ADDR_FROM       => REDIR_UP_ADDR_FROM,
  REDIR_UP_CMD          => REDIR_UP_CMDA,

  REDIR_DOWN_CMD        => REDIR_DOWN_CMD,
  REDIR_DOWN_ADDR_TO    => REDIR_DOWN_ADDR_TO,
  REDIR_DOWN_ADDR_FROM  => REDIR_DOWN_ADDR_FROM,
  REDIR_DOWN_DATA       => REDIR_DOWN_DATA,

  THIS_ADDR          => THIS_ADDR,
  MAS_ADDR       => X"34",
  COPY_PKT       => COPY_PKT,
  PKT_RDY        => PKT_RDY,

  -- DDM
  DDM_PKT_RDY       => DDM_PKT_RDY,
  DDM_PKT_CLR       => DDM_PKT_CLR,
  DDM_ADDR_OUT          => DDM_ADDR_TO_OUT,
  DDM_ADDR_FROM_OUT     =>   DDM_ADDR_FROM_OUT,
  DDM_DATA_OUT          => DDM_DATA_OUT,
  DDM_CMD_OUT       => DDM_CMD_OUT,

  -- Extended data manager
  EXT_PKT_RDY       => EXT_PKT_RDY,
  EXT_PKT_CLR       => EXT_PKT_CLR,
  EXT_ADDR_OUT          => EXT_ADDR_TO_OUT,
  EXT_ADDR_FROM_OUT     =>   EXT_ADDR_FROM_OUT,
  EXT_DATA_OUT          => EXT_DATA_OUT,
  EXT_CMD_OUT       => EXT_CMD_OUT,
  EXT_PKT_ABSORB        => EXT_PKT_ABSORB,

  -- Event Mgr
  EMGR_PKT_RDY          => EMGR_PKT_RDY,
  EMGR_PKT_CLR          => EMGR_PKT_CLR,
  EMGR_CMD_OUT          => EMGR_CMD_OUT,
  EMGR_DATA_OUT         => EMGR_DATA_OUT,

  -- Scheduled Data Mgr
  SCHED_PKT_RDY         => SCHED_PKT_RDY,
  SCHED_PKT_CLR         => SCHED_PKT_CLR,
  SCHED_CMD_OUT         => SCHED_CMD_OUT,
  SCHED_DATA_OUT        => SCHED_DATA_OUT,
```

```
  SCHED_ADDR_TO_OUT     =>  SCHED_ADDR_TO_OUT,

  -- Normal Data Mgr
  NORMAL_PKT_RDY        =>  NORMAL_PKT_RDY,
  NORMAL_PKT_CLR        =>  NORMAL_PKT_CLR,
  NORMAL_CMD_OUT        =>  NORMAL_CMD_OUT,
  NORMAL_DATA_OUT       =>  NORMAL_DATA_OUT,
  NORMAL_ADDR_TO_OUT    =>  NORMAL_ADDR_TO_OUT,

  -- Unscheduled Data Mgr
  UNSCHED_PKT_RDY       =>  UNSCHED_PKT_RDY,
  UNSCHED_PKT_CLR       =>  UNSCHED_PKT_CLR,
  UNSCHED_CMD_OUT       =>  UNSCHED_CMD_OUT,
  UNSCHED_DATA_OUT      =>  UNSCHED_DATA_OUT,
  UNSCHED_ADDR_TO_OUT       =>   UNSCHED_ADDR_TO_OUT,

  DEBUG_CMDPROC         =>  DEBUG_CMDPROC
);
--=====================================================

CDEFRAMER: DEFRAMER port map(
  RESET_L     => BLOCK_RESET_L,
  CLK       => BLOCK_HSCLK,
  RXDATA      => RXDATA1X,
  VLTN      => VLTN,
  LFI_L     => LFI_L,
  RXCMD     => RXCMD1X,
  RXSCD     => RXSCD1X,
  RXCLK     => RXCLK1X,
  BLOCK_PKT_GOOD     => BLOCK_PKT_GOOD,
  REDIR_INX_CMD      => REDIR_IN1_CMD,
-- REDIR_INX_NETTIME   => NET_TIME,           --for
now, do not use this value
  REDIR_INX_ADDR_TO  => REDIR_IN1_ADDR_TO,
  REDIR_INX_ADDR_FROM => REDIR_IN1_ADDR_FROM,
  REDIR_INX_DATA     => REDIR_IN1_DATA,
--  STARTFRAME        => STARTFRAME,
  POSCTR_OUT      => POSCTR_DEF
--  DEBUG_DEF         => DEBUG_DEF
);
```

```
C_DDM2: DDM2 port map(
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L          => BLOCK_RESET_L,
  BLOCK_WRITE_L          => BLOCK_WRITE_L,
  BLOCK_ENABLE           => BLOCK_ENABLE,
  BLOCK_DONE         => BLOCK_DONE,
  THIS_ADDR          => THIS_ADDR,
  BLOCK_PKT_RDY          => DDM_PKT_RDY,
  BLOCK_PKT_CLR          => DDM_PKT_CLR,
  BLOCK_PKT_ADDR_TO_OUT     => DDM_ADDR_TO_OUT,
  BLOCK_PKT_CMD_OUT      => DDM_CMD_OUT,
  BLOCK_PKT_ADDR_FROM_OUT  => DDM_ADDR_FROM_OUT,
  BLOCK_PKT_DATA_OUT         => DDM_DATA_OUT,
  BLOCK_DATA_IN          => BLOCK_DATA_IN,
  BLOCK_ADDR_IN          => BLOCK_ADDR_IN,
  BLOCK_DATA_OUT         => BLOCK_DATA_OUT,

  NETTIME         => NET_TIME,
  BLOCK_PKT_ADDR_TO_IN  => REDIR_UP_ADDR_TO,
  BLOCK_PKT_ADDR_FROM_IN   => REDIR_UP_ADDR_FROM,
  BLOCK_PKT_CMD_IN       => REDIR_UP_CMD,
  BLOCK_PKT_DATA_IN      => REDIR_UP_DATA,
  BLOCK_PKT_GOOD         => BLOCK_PKT_GOOD,

  DEBUG_DDM          => DEBUG_DDM
);

RXCLK1X <= RXCLK1;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    RXDATA1X <= RXDATA1;
    RXCMD1X <= RXCMD1;
    RXSCD1X <= RXSCD1;

    REFCLK1X <= REFCLK1;
    TXFULL_LX <= TXFULL_L;
  end if;
end process;
```

```
--=====================================================

CINDEXMGR: INDEXMGR port map(
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L          => BLOCK_RESET_L,
  BLOCK_PKT_GOOD         => BLOCK_PKT_GOOD,

  PKT_LENGTH         => PKT_LENGTH,
  START_FRAME        => STARTFRAME,
  IN_POSCTR          => POSCTR_DEF,
  OUT_POSCTR         => POSCTR_FRM,
  OPEN_LOOP_MODE         => OPEN_LOOP_MODE_INDEX,
  PKT_ALIGNMENT          => PKT_ALIGNMENT,
  TXCLK       => TXFULL_LX,
  RXCLK       => RXCLK1X
);
--=====================================================

--=====================================================
CEVENT_MAN: event_man port map(
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L          => BLOCK_RESET_L,

  BLOCK_PKT_COMPLETE         => BLOCK_PKT_COMPLETE,
  BLOCK_PKT_GOOD         => MGR_PKT_GOODA,

  BLOCK_CMD_IN           => REDIR_UP_CMD,
  BLOCK_DATA_IN          => REDIR_UP_DATA,

  BLOCK_PKT_RDY          => EMGR_PKT_RDY,
  BLOCK_PKT_CLR          => EMGR_PKT_CLR,
  BLOCK_CMD_OUT          => EMGR_CMD_OUT,
  BLOCK_DATA_OUT         => EMGR_DATA_OUT,

  NET_TIME       => NET_TIME,

  FAULT_INJ          => FAULT_INJ,
  FAULT_SOURCE_L         => FAULT_SOURCE_L,
  FAULT_RESET_L          => FAULT_RESET_L,
  FREEZE      => '0',
  PKT_MODE       => PKT_MODE
);
```

```
CEXTENDED_MGR : EXTENDED_MGR port map(
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L          => BLOCK_RESET_L,
  BLOCK_PKT_GOOD         => BLOCK_PKT_GOOD,

  BLOCK_PKT_RDY          => EXT_PKT_RDY,
  BLOCK_PKT_CLR          => EXT_PKT_CLR,
  BLOCK_PKT_ADDR_TO_OUT => EXT_ADDR_TO_OUT,
  BLOCK_PKT_ADDR_FROM_OUT  => EXT_ADDR_FROM_OUT,
  BLOCK_PKT_CMD_OUT      => EXT_CMD_OUT,
  BLOCK_PKT_DATA_OUT     => EXT_DATA_OUT,

  BLOCK_PKT_CMD_IN       => REDIR_UP_CMD,
  BLOCK_PKT_ADDR_FROM_IN   => REDIR_UP_ADDR_FROM,
  BLOCK_PKT_ADDR_TO_IN  => REDIR_UP_ADDR_TO,
  BLOCK_PKT_DATA_IN      => REDIR_UP_DATA,

  PIN_RST         => PIN_RST,
  BLOCK_PKT_ABSORB       => EXT_PKT_ABSORB,
  SYS_RST_COMM           => SYS_RST_COMM,

  LOAD_CLK_VAL           => LOAD_CLK_VAL,
  NEW_CLK_VAL        => NEW_CLK_VAL,
  THIS_ADDR          => THIS_ADDR
);

--=====================================================
CFRAMER: framer port map(
    RESET_L           => BLOCK_RESET_L,
    CLK        => BLOCK_HSCLK,
    TXDATA         => TXDATA1,
    TXCMD          => TXCMD1,
    TXSCD          => TXSCD1,
    REFCLK         => REFCLK1X,

    REDIR_OUTX_CMD           => REDIR_OUT1_CMD,
    REDIR_OUTX_ADDR_TO       => REDIR_OUT1_ADDR_TO,
    REDIR_OUTX_ADDR_FROM     => REDIR_OUT1_ADDR_FROM,
    REDIR_OUTX_DATA      => REDIR_OUT1_DATA,
    NETTIME              => NET_TIME,
    TXFULL_L             => TXFULL_LX,
```

```
    POSCTR_OUT          => POSCTR_FRM,
    COPY_PKT            => COPY_PKT,
    PKT_RDY          => PKT_RDY,
    STARTFRAME          => STARTFRAME
);
--=======================================================
CSCHED_DM: sched_dm port map(
  BLOCK_HSCLK        => BLOCK_HSCLK,
  BLOCK_RESET_L         => BLOCK_RESET_L,

  BLOCK_PKT_COMPLETE       =>     BLOCK_PKT_COMPLETE,
  BLOCK_PKT_GOOD        => MGR_PKT_GOODA,

  BLOCK_PKT_RDY         => SCHED_PKT_RDY,
  BLOCK_PKT_CLR         => SCHED_PKT_CLR,

  BLOCK_CMD          => REDIR_UP_CMDA,
  BLOCK_DATA_IN         => REDIR_UP_DATA,
  BLOCK_FROM_ADDR_IN    => REDIR_UP_ADDR_FROM,
  BLOCK_TO_ADDR_OUT     => SCHED_ADDR_TO_OUT,

  NET_TIME       => NET_TIME,

  BLOCK_DATA_OUT        => SCHED_DATA_OUT,
  BLOCK_CMD_OUT         => SCHED_CMD_OUT,

  ACTIVE_DATA       => ACTIVE_DATA_SYNC,

  PKT_MODE       => PKT_MODE
);


CNETCLK_MGR: NETCLK_MGR port map(
  BLOCK_HSCLK => BLOCK_HSCLK,
  BLOCK_RESET_L => BLOCK_RESET_L,
  SETCLK => LOAD_CLK_VAL,
  SETVAL => NEW_CLK_VAL,

  INCREMENT => STARTFRAME,

  CURRTIME => NET_TIME
);
```

```
--=======================================================

CNORMAL_DM: NORMAL_DM port map
     (
    BLOCK_HSCLK      => BLOCK_HSCLK,
    BLOCK_RESET_L    => BLOCK_RESET_L,

    BLOCK_PKT_COMPLETE  => BLOCK_PKT_COMPLETE,
    BLOCK_PKT_GOOD      => MGR_PKT_GOOD,

    BLOCK_PKT_RDY    => NORMAL_PKT_RDY,
    BLOCK_PKT_CLR    => NORMAL_PKT_CLR,

    BLOCK_CMD_IN     => REDIR_UP_CMD,
    BLOCK_CMD_OUT    => NORMAL_CMD_OUT,
    BLOCK_TO_ADDR_OUT   => NORMAL_ADDR_TO_OUT,
    BLOCK_FROM_ADDR_IN  => REDIR_UP_ADDR_FROM,
    NET_TIME         => NET_TIME,

    BLOCK_DATA_IN    => REDIR_UP_DATA,
    BLOCK_DATA_OUT       => NORMAL_DATA_OUT,

    ACTIVE_DATA      => ACTIVE_DATA_NORMAL,

    PKT_MODE         => PKT_MODE,

    --Status/Sensor Information
    STATUS_DATA      => STATUS_DATA,

    SYNC      => SYNC

   );
--=======================================================
CREDIRECTOR: redirector port map(
  REDIR_OUT1_CMD        => REDIR_OUT1_CMD,
  REDIR_OUT1_ADDR_TO       => REDIR_OUT1_ADDR_TO,
  REDIR_OUT1_ADDR_FROM  => REDIR_OUT1_ADDR_FROM,
  REDIR_OUT1_DATA       => REDIR_OUT1_DATA,

  REDIR_IN1_CMD         => REDIR_IN1_CMD,
  REDIR_IN1_ADDR_TO     => REDIR_IN1_ADDR_TO,
```

```
  REDIR_IN1_ADDR_FROM   => REDIR_IN1_ADDR_FROM,
  REDIR_IN1_DATA        => REDIR_IN1_DATA,

--Data going up to command processor
  REDIR_OUT_CMD         => REDIR_UP_CMD,
  REDIR_OUT_ADDR_TO     => REDIR_UP_ADDR_TO,
  REDIR_OUT_ADDR_FROM   => REDIR_UP_ADDR_FROM,
  REDIR_OUT_DATA        => REDIR_UP_DATA,

  REDIR_IN_CMD          => REDIR_DOWN_CMD,
  REDIR_IN_ADDR_TO      => REDIR_DOWN_ADDR_TO,
  REDIR_IN_ADDR_FROM       => REDIR_DOWN_ADDR_FROM,
  REDIR_IN_DATA         => REDIR_DOWN_DATA

);

NET_TIME_OUT <= NET_TIME;

--=====================================================
DEBUG_AUX(23) <= BLOCK_PKT_GOOD;
DEBUG_AUX(22) <= PKT_RDY;
DEBUG_AUX(21) <= DDM_PKT_CLR;
DEBUG_AUX(20) <= DDM_PKT_RDY;
DEBUG_AUX( 19 downto 12 ) <= DEBUG_CMDPROC;
DEBUG_AUX( 11 downto 8 ) <= DDM_CMD_OUT;
DEBUG_AUX( 7 ) <= DDM_PKT_CLR;
DEBUG_AUX( 6 )  <= DDM_PKT_RDY;
DEBUG_AUX( 5 ) <= COPY_PKT;
DEBUG_AUX( 4 downto 1 ) <= REDIR_DOWN_CMD;

DEBUG_AUX(31 downto 24 ) <= THIS_ADDR;

process( BLOCK_RESET_L, BLOCK_HSCLK )
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     DEBUG_DEF <= X"00";
     DEBUG_FRM <= X"00";
    else
     DEBUG_DEF(7 downto 0) <= DEBUG_DDM( 15 downto 8);
     DEBUG_FRM(7 downto 0) <= DEBUG_DDM( 7 downto 0 );
    end if;
      end if;
end process;

end Behavioral;
```

9.2.5 Asynchronous_DM.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity asynchronous_dm is
    Port (
    BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_GOOD: in std_logic;

    BLOCK_PKT_RDY: out std_logic;
    BLOCK_PKT_CLR: in std_logic;

    NET_TIME: in T_NETTIME;

    BLOCK_PKT_CMD_IN: in T_COMMAND;
    BLOCK_PKT_FROM_ADDR_IN: in T_NODEADDR;
    BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;

    BLOCK_PKT_CMD_OUT: out T_COMMAND;
    BLOCK_PKT_TO_ADDR_OUT: out T_NODEADDR;
    BLOCK_PKT_DATA_OUT: out T_DATA_PAYLOAD;

    ASYNC_DATA_OUT: out T_ASYNC_ATTR_ARRAY;
    ASYNC_DATA_IN: in T_ASYNC_ATTR_ARRAY;
    ASYNC_DATA_DIR: in std_logic_vector( C_ASYNC_DATA_MAX
downto 0 )  -- 1 = data comes from packet, 0 = device
    );
end asynchronous_dm;

architecture Behavioral of asynchronous_dm is
  component PIPE_MUX16 is
    port(
     BLOCK_HSCLK: in std_logic;
     DATA_OUT: out std_logic_vector( 15 downto 0 );
     INDEX: in std_logic_vector( 3 downto 0 );
     D0: in std_logic_vector( 15 downto 0 );
     D1: in std_logic_vector( 15 downto 0 );
     D2: in std_logic_vector( 15 downto 0 );
     D3: in std_logic_vector( 15 downto 0 );
     D4: in std_logic_vector( 15 downto 0 );
     D5: in std_logic_vector( 15 downto 0 );
     D6: in std_logic_vector( 15 downto 0 );
     D7: in std_logic_vector( 15 downto 0 );
     D8: in std_logic_vector( 15 downto 0 );
     D9: in std_logic_vector( 15 downto 0 );
     DA: in std_logic_vector( 15 downto 0 );
     DB: in std_logic_vector( 15 downto 0 );
     DC: in std_logic_vector( 15 downto 0 );
     DD: in std_logic_vector( 15 downto 0 );
     DE: in std_logic_vector( 15 downto 0 );
     DF: in std_logic_vector( 15 downto 0 )
    );
  end component;

  type T_FSM_SCHED is (idle, ready, waitclr );
  signal STATE, NEXT_STATE: T_FSM_SCHED;

  constant C_ASYNC_ADDR_WIDTH: integer := 4;

  --per cell signals
  signal ADDR_MATCH1, ADDR_MATCH2, ADDR_MATCH3,
ADDR_MATCH_ANY: std_logic_vector( C_ASYNC_DATA_MAX
downto 0 );
  signal WR_EN, WR_EN_LOCKOUT: std_logic_vector(
C_ASYNC_DATA_MAX downto 0 );
```

```
  signal GLOBAL_WR_EN: std_logic;

  signal ASYNC_DATA, ASYNC_DATA_MR: T_ASYNC_ATTR_ARRAY;

  signal CMD_SET_ASYNC, CMD_GET_ASYNC: std_logic;

  signal MUX_D0,MUX_D1,MUX_D2,MUX_D3: std_logic_vector(
15 downto 0 );
  signal MUX_D4,MUX_D5,MUX_D6,MUX_D7: std_logic_vector(
15 downto 0 );
  signal MUX_D8,MUX_D9,MUX_DA,MUX_DB: std_logic_vector(
15 downto 0 );
  signal MUX_DC,MUX_DD,MUX_DE,MUX_DF: std_logic_vector(
15 downto 0 );

  signal MUX_C1, MUX_C2, MUX_C3: std_logic_vector( 15
downto 0 );
  signal MUX_CODE: std_logic_vector( 15 downto 0 );

begin

ASYNC_DATA_OUT <= ASYNC_DATA;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     GLOBAL_WR_EN <= '0';
     CMD_SET_ASYNC <= '0';
     CMD_GET_ASYNC <= '0';

    else
     if( CMD_SET_ASYNC = '1' and BLOCK_PKT_GOOD = '1')
then
       GLOBAL_WR_EN <= '1';
     else
       GLOBAL_WR_EN <= '0';
     end if;

     --Set if the command in is to set asynchronous data
     if( BLOCK_PKT_CMD_IN = C_CMD_SETVAL_ASYNC ) then
       CMD_SET_ASYNC <= '1';
     else
       CMD_SET_ASYNC <= '0';
     end if;

     --Set if the command in is to set asynchronous data
     if( BLOCK_PKT_CMD_IN = C_CMD_GETVAL_ASYNC ) then
       CMD_GET_ASYNC <= '1';
     else
       CMD_GET_ASYNC <= '0';
     end if;

    end if;
  end if;
end process;

G1: for i in 0 to 7 generate --C_ASYNC_DATA_MAX generate
  process( BLOCK_HSCLK )  is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     --an address match consists of a matching addess, as
well as the address enable flag set
     if( BLOCK_PKT_DATA_IN(1)(C_ASYNC_ADDR_WIDTH - 1
downto 0) = CONV_STD_LOGIC_VECTOR(i,C_ASYNC_ADDR_WIDTH)
and BLOCK_PKT_DATA_IN(1)(7) = '1' ) then
       ADDR_MATCH1(i) <= '1';
     else
       ADDR_MATCH1(i) <= '0';
     end if;

     --an address match consists of a matching addess, as
well as the address enable flag set
     if( BLOCK_PKT_DATA_IN(4)(C_ASYNC_ADDR_WIDTH-1 downto
0) = CONV_STD_LOGIC_VECTOR(i,C_ASYNC_ADDR_WIDTH) and
BLOCK_PKT_DATA_IN(4)(7) = '1') then
       ADDR_MATCH2(i) <= '1';
     else
       ADDR_MATCH2(i) <= '0';
     end if;

     --an address match consists of a matching addess, as
well as the address enable flag set
```

```
    if( BLOCK_PKT_DATA_IN(7)(C_ASYNC_ADDR_WIDTH-1 downto
0) = CONV_STD_LOGIC_VECTOR(i,C_ASYNC_ADDR_WIDTH) and
BLOCK_PKT_DATA_IN(7)(7) = '1') then
      ADDR_MATCH3(i) <= '1';
    else
      ADDR_MATCH3(i) <= '0';
    end if;

    ADDR_MATCH_ANY(i) <= ADDR_MATCH1(i) or
ADDR_MATCH2(i) or ADDR_MATCH3(i);

    --enable write if there is an address match, the
data direction is good, and the global write enable is
set
    WR_EN(i) <= ADDR_MATCH_ANY(i) and ASYNC_DATA_DIR(i)
and GLOBAL_WR_EN;


   end if;
  end process;

  process( ADDR_MATCH1(i), ADDR_MATCH2(i),
ADDR_MATCH3(i), BLOCK_PKT_DATA_IN ) is
  begin
    if( ADDR_MATCH1(i) = '1' ) then
      ASYNC_DATA_MR(i) <= BLOCK_PKT_DATA_IN(2) &
BLOCK_PKT_DATA_IN(3);
    elsif( ADDR_MATCH2(i) = '1' ) then
      ASYNC_DATA_MR(i) <= BLOCK_PKT_DATA_IN(5) &
BLOCK_PKT_DATA_IN(6);
    elsif( ADDR_MATCH3(i) = '1' ) then
      ASYNC_DATA_MR(i) <= BLOCK_PKT_DATA_IN(8) &
BLOCK_PKT_DATA_IN(9);
    else
      ASYNC_DATA_MR(i) <= X"7E7E";
    end if;
  end process;

  process( BLOCK_HSCLK ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' ) then
       ASYNC_DATA(i) <= ASYNC_DATA_IN(i);
       WR_EN_LOCKOUT(i) <= '0';
     else
       if( ASYNC_DATA_DIR(i) = '0' ) then
         ASYNC_DATA(i) <= ASYNC_DATA_IN(i);
       elsif( WR_EN(i) = '1' and WR_EN_LOCKOUT(i) = '0'
) then
         WR_EN_LOCKOUT(i) <= '1';
         --copy the data from the mux result
         ASYNC_DATA(i) <= ASYNC_DATA_MR(i);
       elsif( WR_EN(i) = '0' ) then
         WR_EN_LOCKOUT(i) <= '0';
       end if;
     end if;
    end if;
  end process;



end generate;


G2: for i in 8 to 15 generate
  WR_EN_LOCKOUT(i) <= '0';
  ASYNC_DATA(i) <= X"0000";
  ADDR_MATCH1(i) <= '0';
  ADDR_MATCH2(i) <= '0';
  ADDR_MATCH3(i) <= '0';
end generate;


process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( CMD_GET_ASYNC = '1' and BLOCK_PKT_GOOD = '1' )
then
     BLOCK_PKT_CMD_OUT      <= C_CMD_GETVAL_ASYNC;
     BLOCK_PKT_TO_ADDR_OUT  <= BLOCK_PKT_FROM_ADDR_IN;

     BLOCK_PKT_DATA_OUT(0)  <= X"01";
     BLOCK_PKT_DATA_OUT(1)  <= BLOCK_PKT_DATA_IN(1);
     BLOCK_PKT_DATA_OUT(2)  <= MUX_C1(15 downto 8);
```

```
    BLOCK_PKT_DATA_OUT(3) <= MUX_C1(7 downto 0);
    BLOCK_PKT_DATA_OUT(4) <= BLOCK_PKT_DATA_IN(4);
    BLOCK_PKT_DATA_OUT(5) <= MUX_C2(15 downto 8);
    BLOCK_PKT_DATA_OUT(6) <= MUX_C2(7 downto 0);
    BLOCK_PKT_DATA_OUT(7) <= BLOCK_PKT_DATA_IN(7);
    BLOCK_PKT_DATA_OUT(8) <= MUX_C3(15 downto 8);
    BLOCK_PKT_DATA_OUT(9) <= MUX_C3(7 downto 0);
   end if;
  end if;
end process;


MUX_CODE <= "0000" & BLOCK_PKT_DATA_IN(1)(3 downto 0 ) &
BLOCK_PKT_DATA_IN(4)(3 downto 0 ) &
BLOCK_PKT_DATA_IN(7)(3 downto 0 );
CPIPE_MUX16_1: PIPE_MUX16 port map(
   BLOCK_HSCLK  => BLOCK_HSCLK,
   DATA_OUT     => MUX_C3,
   INDEX   => MUX_CODE(3 downto 0 ),
   D0 => MUX_D0,
   D1 => MUX_D1,
   D2 => MUX_D2,
   D3 => MUX_D3,
   D4 => MUX_D4,
   D5 => MUX_D5,
   D6 => MUX_D6,
   D7 => MUX_D7,
   D8 => MUX_D8,
   D9 => MUX_D9,
   DA => MUX_DA,
   DB => MUX_DB,
   DC => MUX_DC,
   DD => MUX_DD,
   DE => MUX_DE,
   DF => MUX_DF
  );

CPIPE_MUX16_2: PIPE_MUX16 port map(
   BLOCK_HSCLK  => BLOCK_HSCLK,
   DATA_OUT     => MUX_C2,
   INDEX   => MUX_CODE(7 downto 4 ),
   D0 => MUX_D0,
   D1 => MUX_D1,
   D2 => MUX_D2,
   D3 => MUX_D3,
   D4 => MUX_D4,
   D5 => MUX_D5,
   D6 => MUX_D6,
   D7 => MUX_D7,
   D8 => MUX_D8,
   D9 => MUX_D9,
   DA => MUX_DA,
   DB => MUX_DB,
   DC => MUX_DC,
   DD => MUX_DD,
   DE => MUX_DE,
   DF => MUX_DF
  );

CPIPE_MUX16_3: PIPE_MUX16 port map(
   BLOCK_HSCLK  => BLOCK_HSCLK,
   DATA_OUT     => MUX_C1,
   INDEX   => MUX_CODE(11 downto 8 ),
   D0 => MUX_D0,
   D1 => MUX_D1,
   D2 => MUX_D2,
   D3 => MUX_D3,
   D4 => MUX_D4,
   D5 => MUX_D5,
   D6 => MUX_D6,
   D7 => MUX_D7,
   D8 => MUX_D8,
   D9 => MUX_D9,
   DA => MUX_DA,
   DB => MUX_DB,
   DC => MUX_DC,
   DD => MUX_DD,
   DE => MUX_DE,
   DF => MUX_DF
  );

MUX_D0 <= ASYNC_DATA(0);
MUX_D1 <= ASYNC_DATA(1);
MUX_D2 <= ASYNC_DATA(2);
```

```
MUX_D3 <= ASYNC_DATA(3);
MUX_D4 <= ASYNC_DATA(4);
MUX_D5 <= ASYNC_DATA(5);
MUX_D6 <= ASYNC_DATA(6);
MUX_D7 <= ASYNC_DATA(7);
MUX_D8 <= ASYNC_DATA(8);
MUX_D9 <= ASYNC_DATA(9);
MUX_DA <= ASYNC_DATA(10);
MUX_DB <= ASYNC_DATA(11);
MUX_DC <= ASYNC_DATA(12);
MUX_DD <= ASYNC_DATA(13);
MUX_DE <= ASYNC_DATA(14);
MUX_DF <= ASYNC_DATA(15);


process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXT_STATE;
    end if;
  end if;
end process;

process( STATE, BLOCK_PKT_GOOD, BLOCK_PKT_DATA_IN,
CMD_GET_ASYNC, BLOCK_PKT_CLR ) is
begin
  case( STATE ) is
  when idle =>
    BLOCK_PKT_RDY <= '0';
    --wait for a normal packet that is destined for this
node
    if( BLOCK_PKT_GOOD = '1' and CMD_GET_ASYNC = '1' and
BLOCK_PKT_DATA_IN(0)(0) = '0') then
     NEXT_STATE <= ready;
    else
     NEXT_STATE <= idle;
    end if;
  when ready =>
    BLOCK_PKT_RDY <= '1';
    --wait for clr to go high, indicating packet is
absorbed
    if( BLOCK_PKT_CLR = '1' ) then
     NEXT_STATE <= waitclr;
    else
     NEXT_STATE <= ready;
    end if;
  when waitclr =>
    BLOCK_PKT_RDY <= '0';
    --wait for clear to fall back to 0 and for the packet
to end
    --wait for block_pkt_clr to fall to prevent re-entry
    if( BLOCK_PKT_CLR = '0' and BLOCK_PKT_GOOD = '0')
then
     NEXT_STATE <= idle;
    else
     NEXT_STATE <= waitclr;
    end if;
  when others =>
    BLOCK_PKT_RDY <= '0';
    NEXT_STATE <= idle;
  end case;
end process;


end Behavioral;
```

9.2.6 PIPE_MUX16.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PIPE_MUX16 is
  port(
    BLOCK_HSCLK: in std_logic;
    DATA_OUT: out std_logic_vector( 15 downto 0 );
    INDEX: in std_logic_vector( 3 downto 0 );
    D0: in std_logic_vector( 15 downto 0 );
    D1: in std_logic_vector( 15 downto 0 );
    D2: in std_logic_vector( 15 downto 0 );
    D3: in std_logic_vector( 15 downto 0 );
    D4: in std_logic_vector( 15 downto 0 );
    D5: in std_logic_vector( 15 downto 0 );
    D6: in std_logic_vector( 15 downto 0 );
    D7: in std_logic_vector( 15 downto 0 );
    D8: in std_logic_vector( 15 downto 0 );
    D9: in std_logic_vector( 15 downto 0 );
    DA: in std_logic_vector( 15 downto 0 );
    DB: in std_logic_vector( 15 downto 0 );
    DC: in std_logic_vector( 15 downto 0 );
    DD: in std_logic_vector( 15 downto 0 );
    DE: in std_logic_vector( 15 downto 0 );
    DF: in std_logic_vector( 15 downto 0 )
  );
end PIPE_MUX16;

architecture Behavioral of PIPE_MUX16 is
signal SEL00, SEL01, SEL10, SEL11: std_logic_vector( 15
downto 0 );
begin

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) )then
    case  INDEX(1 downto 0) is
     when "00" => SEL00 <= D0;
            SEL01 <= D4;
            SEL10 <= D8;
            SEL11 <= DC;
     when "01" => SEL00 <= D1;
            SEL01 <= D5;
            SEL10 <= D9;
            SEL11 <= DD;
     when "10" => SEL00 <= D2;
            SEL01 <= D6;
            SEL10 <= DA;
            SEL11 <= DE;
     when "11" => SEL00 <= D3;
            SEL01 <= D7;
            SEL10 <= DB;
            SEL11 <= DF;
     when others => SEL00 <= X"ABCD";
    end case;

    case  INDEX(3 downto 2) is
     when "00" => DATA_OUT <= SEL00;
     when "01" => DATA_OUT <= SEL01;
     when "10" => DATA_OUT <= SEL10;
     when "11" => DATA_OUT <= SEL11;
     when others => SEL00 <= X"ABCD";
    end case;

  end if;
end process;
end Behavioral;
```

9.2.7 CMDProc.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity cmdproc is
    Port (
    BLOCK_RESET_L: in std_logic;
    BLOCK_HSCLK: in std_logic;

    --packet sent to redirector
    REDIR_DOWN_DATA: out T_DATA_PAYLOAD;
    REDIR_DOWN_ADDR_TO:  out T_NODEADDR;
    REDIR_DOWN_ADDR_FROM: out T_NODEADDR;
    REDIR_DOWN_CMD: out T_COMMAND;

    --packet coming in from redirector
    REDIR_UP_CMD: in T_COMMAND;
    REDIR_UP_ADDR_TO:  in T_NODEADDR;
    REDIR_UP_ADDR_FROM: in T_NODEADDR;
    REDIR_UP_DATA: in T_DATA_PAYLOAD;

    -- Environmental info
    THIS_ADDR:       in T_NODEADDR;
    MAS_ADDR:    in T_NODEADDR;

    -- LATCH_DATA:   out std_logic;
    -- Info on packet status
    BLOCK_PKT_GOOD: in std_logic;
    MGR_PKT_GOOD: out std_logic;

    -- Event Mgr
    EMGR_PKT_RDY: in std_logic;
    EMGR_PKT_CLR: out std_logic;
    EMGR_CMD_OUT: in T_COMMAND;
    EMGR_DATA_OUT: in T_DATA_PAYLOAD;

    -- Scheduled Data Mgr
    SCHED_PKT_RDY: in std_logic;
    SCHED_PKT_CLR: out std_logic;
    SCHED_CMD_OUT: in T_COMMAND;
    SCHED_DATA_OUT: in T_DATA_PAYLOAD;
    SCHED_ADDR_TO_OUT:  in T_NODEADDR;

    -- Normal Data Mgr
    NORMAL_PKT_RDY: in std_logic;
    NORMAL_PKT_CLR: out std_logic;
    NORMAL_CMD_OUT: in T_COMMAND;
    NORMAL_DATA_OUT: in T_DATA_PAYLOAD;
    NORMAL_ADDR_TO_OUT: in T_NODEADDR;

    -- Unscheduled Data Mgr
    UNSCHED_PKT_RDY: in std_logic;
    UNSCHED_PKT_CLR: out std_logic;
    UNSCHED_CMD_OUT: in T_COMMAND;
    UNSCHED_DATA_OUT: in T_DATA_PAYLOAD;
    UNSCHED_ADDR_TO_OUT: in T_NODEADDR;

    -- DDM
    DDM_PKT_RDY: in std_logic;
    DDM_PKT_CLR: out std_logic;
    DDM_CMD_OUT: in T_COMMAND;
    DDM_ADDR_OUT: in T_NODEADDR;
    DDM_DATA_OUT: in T_DATA_PAYLOAD;
    DDM_ADDR_FROM_OUT: in T_NODEADDR;

    --Extended data manage
    EXT_PKT_RDY: in std_logic;
    EXT_PKT_CLR: out std_logic;
    EXT_CMD_OUT: in T_COMMAND;
    EXT_ADDR_OUT: in T_NODEADDR;
    EXT_DATA_OUT: in T_DATA_PAYLOAD;
```

```
    EXT_ADDR_FROM_OUT: in T_NODEADDR;
    EXT_PKT_ABSORB: in std_logic;

    COPY_PKT: in std_logic;
    PKT_RDY: out std_logic;

    DEBUG_CMDPROC: out std_logic_vector( 7 downto 0 )
   );
end cmdproc;

architecture Behavioral of cmdproc is
  type T_OPMODE is ( FORWARD, NORMAL, SCHED, UNSCHED,
EVENT, NULLPKT, DDM, EXTENDED );
  signal OUTMODE, OUTMODE2 : T_OPMODE;

  type T_STATE is ( RELEASE, LOCK_CHANGES, COPY_ACK );
  signal STATE, NEXTSTATE: T_STATE;

  signal DEST_THIS: std_logic;  --this is the
destination
  signal ADDR_MATCH: std_logic;
  signal NULL_MATCH: std_logic;
  signal FROM_THIS: std_logic;

    --packet sent to redirector
  signal REDIR_DOWN_DATAX: T_DATA_PAYLOAD;
  signal REDIR_DOWN_ADDR_TOX:  T_NODEADDR;
  signal REDIR_DOWN_ADDR_FROMX: T_NODEADDR;
  signal REDIR_DOWN_CMDX: T_COMMAND;

  signal REDIR_DOWN_DATA_MUX1: T_DATA_PAYLOAD;
  signal REDIR_DOWN_ADDR_TO_MUX1:  T_NODEADDR;
  signal REDIR_DOWN_ADDR_FROM_MUX1: T_NODEADDR;
  signal REDIR_DOWN_CMD_MUX1: T_COMMAND;

  signal REDIR_DOWN_DATA_MUX2: T_DATA_PAYLOAD;
  signal REDIR_DOWN_ADDR_TO_MUX2:  T_NODEADDR;
  signal REDIR_DOWN_ADDR_FROM_MUX2: T_NODEADDR;
  signal REDIR_DOWN_CMD_MUX2: T_COMMAND;


  signal LATCH_DATAX: std_logic;
  signal FORWARD_RDY: std_logic;
  signal FORWARD_CLR: std_logic;

    --packet sent to redirector
  signal FORWARD_DATAX: T_DATA_PAYLOAD;
  signal FORWARD_ADDR_TOX:  T_NODEADDR;
  signal FORWARD_ADDR_FROMX: T_NODEADDR;
  signal FORWARD_CMDX: T_COMMAND;
  signal FORWARD_LOCKOUT: std_logic;
  signal FORWARD_PKT: std_logic;
  signal FORWARD_CLRX: std_logic;
begin
  process( BLOCK_HSCLK, BLOCK_RESET_L ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( REDIR_UP_ADDR_FROM = THIS_ADDR ) then
       FROM_THIS <= '1';
     else
       FROM_THIS <= '0';
     end if;
    end if;
  end process;

  process( BLOCK_HSCLK, BLOCK_RESET_L ) is
  begin
    if( rising_edge( BLOCK_HSCLK ))  then
     FORWARD_CLRX <= FORWARD_CLR;

     if( BLOCK_RESET_L = '0' ) then
       FORWARD_RDY <= '0';
       FORWARD_PKT <= '0';
     else
       if( NULL_MATCH = '0' and ADDR_MATCH = '0' and
FROM_THIS = '0') then
         FORWARD_PKT <= '1';
       else
         FORWARD_PKT <= '0';
       end if;

       if( FORWARD_PKT = '1' and FORWARD_CLRX = '0' and
BLOCK_PKT_GOOD = '1' )  then
```

```
        --packet sent to redirector
        FORWARD_DATAX      <= REDIR_UP_DATA;
        FORWARD_ADDR_TOX   <= REDIR_UP_ADDR_TO;
        FORWARD_ADDR_FROMX       <= REDIR_UP_ADDR_FROM;
        FORWARD_CMDX       <= REDIR_UP_CMD;

        FORWARD_RDY <= '1';
       elsif( FORWARD_CLRX = '1' ) then
        FORWARD_RDY <= '0';
       end if;
     end if;
    end if;
  end process;


  --Determine if this packet is addressed to this node
  --This process will determine if the data block is
good.  If so, it will
  --forward the good message to the neighboring nodes to
indicate that
  --the nodes can operate on the valid data.  If the
packet is for a node
  --with a different address, then the good signal never
is passed on.
  process( BLOCK_HSCLK, BLOCK_RESET_L ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' ) then
       MGR_PKT_GOOD <= '0';
       DEST_THIS <= '0';
     else
       if( ADDR_MATCH = '1'  or NULL_MATCH = '1' ) then
         MGR_PKT_GOOD <= BLOCK_PKT_GOOD;   --pass
pkt_good to managers
         DEST_THIS <= '1';  --indicate this is the
destination node
       else
         MGR_PKT_GOOD <= '0';  --block pkt_good
         DEST_THIS <= '0';  --indicate this is not
destination node
       end if;
     end if;
    end if;
  end process;

  process( BLOCK_HSCLK, BLOCK_RESET_L ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' ) then
       ADDR_MATCH <= '0';
       NULL_MATCH <= '0';
     else
       if( THIS_ADDR = REDIR_UP_ADDR_TO  ) then
         ADDR_MATCH <= '1';  --indicate this is the
destination node
       else
         ADDR_MATCH <= '0';  --block pkt_good
       end if;

       if( REDIR_UP_CMD = C_CMD_NULL ) then
         NULL_MATCH <= '1';
       else
         NULL_MATCH <= '0';
       end if;
     end if;
    end if;
  end process;

  --Select output data to transmit based on priority
  process( BLOCK_HSCLK, BLOCK_PKT_GOOD ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if ( BLOCK_RESET_L = '0' ) then
       --send null packet
       OUTMODE <= NULLPKT;
     else
       --construct priority encoder
       if( COPY_PKT = '0' ) then
         if( EXT_PKT_RDY = '1' ) then
           OUTMODE <= EXTENDED;
         elsif( FORWARD_RDY = '1' and EXT_PKT_ABSORB =
'0' ) then
           OUTMODE <= FORWARD;  --still need to not
forward bad packets
```

```
      elsif( NORMAL_PKT_RDY = '1' ) then
        OUTMODE <= NORMAL;
      elsif( SCHED_PKT_RDY = '1' ) then
        OUTMODE <= SCHED;
      elsif( UNSCHED_PKT_RDY = '1' ) then
        OUTMODE <= UNSCHED;
      elsif( EMGR_PKT_RDY = '1' ) then
        OUTMODE <= EVENT;
      elsif( DDM_PKT_RDY = '1' ) then
        OUTMODE <= DDM;
      else
        OUTMODE <= NULLPKT;
      end if;
     end if;
   end if;
  end if;
end process;


process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK )) then
   --Pipe mux first stage
   case( OUTMODE ) is
     when FORWARD =>
       REDIR_DOWN_DATA_MUX1     <= FORWARD_DATAX;
       REDIR_DOWN_ADDR_TO_MUX1  <= FORWARD_ADDR_TOX;
       REDIR_DOWN_ADDR_FROM_MUX1<= FORWARD_ADDR_FROMX;
       REDIR_DOWN_CMD_MUX1      <= FORWARD_CMDX;
     when NORMAL =>
       REDIR_DOWN_DATA_MUX1     <= NORMAL_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX1  <= NORMAL_ADDR_TO_OUT;
       REDIR_DOWN_ADDR_FROM_MUX1<= THIS_ADDR;
       REDIR_DOWN_CMD_MUX1      <= NORMAL_CMD_OUT;
     when SCHED =>
       REDIR_DOWN_DATA_MUX1     <= SCHED_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX1  <= SCHED_ADDR_TO_OUT;
       REDIR_DOWN_ADDR_FROM_MUX1<= THIS_ADDR;
       REDIR_DOWN_CMD_MUX1      <= SCHED_CMD_OUT;
     when UNSCHED =>
       REDIR_DOWN_DATA_MUX1     <= UNSCHED_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX1  <= UNSCHED_ADDR_TO_OUT;
       REDIR_DOWN_ADDR_FROM_MUX1<= THIS_ADDR;
       REDIR_DOWN_CMD_MUX1      <= UNSCHED_CMD_OUT;
     when others =>
       REDIR_DOWN_DATA_MUX1     <= (X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00"
);
       REDIR_DOWN_ADDR_TO_MUX1         <= X"00";
       REDIR_DOWN_ADDR_FROM_MUX1       <= X"00";
       REDIR_DOWN_CMD_MUX1             <= X"0";

   end case;

   --pipe mux first stage, part 2
   case( OUTMODE ) is
     when EVENT =>
       REDIR_DOWN_DATA_MUX2     <= EMGR_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX2  <= MAS_ADDR;
       REDIR_DOWN_ADDR_FROM_MUX2<= THIS_ADDR;
       REDIR_DOWN_CMD_MUX2      <= EMGR_CMD_OUT;

     when DDM =>
       REDIR_DOWN_DATA_MUX2     <= DDM_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX2  <= DDM_ADDR_OUT;
       REDIR_DOWN_ADDR_FROM_MUX2<= DDM_ADDR_FROM_OUT;
       REDIR_DOWN_CMD_MUX2      <= DDM_CMD_OUT;

     when EXTENDED =>
       REDIR_DOWN_DATA_MUX2     <= EXT_DATA_OUT;
       REDIR_DOWN_ADDR_TO_MUX2  <= EXT_ADDR_OUT;
       REDIR_DOWN_ADDR_FROM_MUX2<= EXT_ADDR_FROM_OUT;
       REDIR_DOWN_CMD_MUX2      <= EXT_CMD_OUT;

     when NULLPKT =>
     --send null packet
       REDIR_DOWN_CMD_MUX2             <= C_CMD_NULL;
       REDIR_DOWN_ADDR_TO_MUX2         <= X"00";
       REDIR_DOWN_ADDR_FROM_MUX2       <= THIS_ADDR;
       REDIR_DOWN_DATA_MUX2            <= (X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"00" );
     when others =>
```

```
        REDIR_DOWN_DATA_MUX2            <= (X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"00" );
        REDIR_DOWN_ADDR_TO_MUX2         <= X"00";
        REDIR_DOWN_ADDR_FROM_MUX2       <= X"00";
        REDIR_DOWN_CMD_MUX2             <= X"0";
    end case;
   end if;
  end process;

  --Forward data to output from selected location as
determined
  --by the variable outmode
  process( BLOCK_HSCLK, BLOCK_RESET_L ) is
  begin
    --pipe mux second stage
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( STATE = RELEASE ) then
       OUTMODE2 <= OUTMODE;
       REDIR_DOWN_CMDX     <= REDIR_DOWN_CMD_MUX1
  or REDIR_DOWN_CMD_MUX2;
       REDIR_DOWN_ADDR_TOX        <=
REDIR_DOWN_ADDR_TO_MUX1    or REDIR_DOWN_ADDR_TO_MUX2;
       REDIR_DOWN_ADDR_FROMX      <=
REDIR_DOWN_ADDR_FROM_MUX1  or REDIR_DOWN_ADDR_FROM_MUX2;
       REDIR_DOWN_DATAX(0)        <=
REDIR_DOWN_DATA_MUX1(0) or REDIR_DOWN_DATA_MUX2(0);
       REDIR_DOWN_DATAX(1)        <=
REDIR_DOWN_DATA_MUX1(1) or REDIR_DOWN_DATA_MUX2(1);
       REDIR_DOWN_DATAX(2)        <=
REDIR_DOWN_DATA_MUX1(2) or REDIR_DOWN_DATA_MUX2(2);
       REDIR_DOWN_DATAX(3)        <=
REDIR_DOWN_DATA_MUX1(3) or REDIR_DOWN_DATA_MUX2(3);
       REDIR_DOWN_DATAX(4)        <=
REDIR_DOWN_DATA_MUX1(4) or REDIR_DOWN_DATA_MUX2(4);
       REDIR_DOWN_DATAX(5)        <=
REDIR_DOWN_DATA_MUX1(5) or REDIR_DOWN_DATA_MUX2(5);
       REDIR_DOWN_DATAX(6)        <=
REDIR_DOWN_DATA_MUX1(6) or REDIR_DOWN_DATA_MUX2(6);
       REDIR_DOWN_DATAX(7)        <=
REDIR_DOWN_DATA_MUX1(7) or REDIR_DOWN_DATA_MUX2(7);
       REDIR_DOWN_DATAX(8)        <=
REDIR_DOWN_DATA_MUX1(8) or REDIR_DOWN_DATA_MUX2(8);
       REDIR_DOWN_DATAX(9)        <=
REDIR_DOWN_DATA_MUX1(9) or REDIR_DOWN_DATA_MUX2(9);

     end if;
    end if;
  end process;

process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     STATE <= RELEASE;
    else
     STATE <= NEXTSTATE;
    end if;
  end if;
end process;


process( OUTMODE2,  STATE, COPY_PKT, OUTMODE ) is
begin
  case( STATE ) is
  when RELEASE =>  --allow prioritization to occur
freely
    LATCH_DATAX <= '0';
    DDM_PKT_CLR <= '0';
    EXT_PKT_CLR <= '0';
    FORWARD_CLR <= '0';
    NORMAL_PKT_CLR <= '0';
    SCHED_PKT_CLR <= '0';
    UNSCHED_PKT_CLR <= '0';

    if( COPY_PKT = '1' ) then
     NEXTSTATE <= LOCK_CHANGES;
    else
     NEXTSTATE <= RELEASE;
    end if;
    PKT_RDY <= '0';
  when LOCK_CHANGES => --freeze prioritization while it
is sampled.
```

```
    LATCH_DATAX <= '1';
    DDM_PKT_CLR <= '0';
    EXT_PKT_CLR <= '0';
    FORWARD_CLR <= '0';
    NORMAL_PKT_CLR <= '0';
    SCHED_PKT_CLR <= '0';
    UNSCHED_PKT_CLR <= '0';

    if( COPY_PKT = '0' ) then
     NEXTSTATE <= COPY_ACK;
    else
     NEXTSTATE <= LOCK_CHANGES;
    end if;

    PKT_RDY <= '1';
  when COPY_ACK =>       --send acknowledgement to clear
ready signal
    LATCH_DATAX <= '1';
    case( OUTMODE2 ) is
     when EXTENDED =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '1';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
     when FORWARD =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '1';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
     when NORMAL =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '1';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
     when SCHED =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '1';
        UNSCHED_PKT_CLR <= '0';
     when UNSCHED =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '1';
     when DDM =>
        DDM_PKT_CLR <= '1';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
     when NULLPKT =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
     when others =>
        DDM_PKT_CLR <= '0';
        EXT_PKT_CLR <= '0';
        FORWARD_CLR <= '0';
        NORMAL_PKT_CLR <= '0';
        SCHED_PKT_CLR <= '0';
        UNSCHED_PKT_CLR <= '0';
    end case;
    NEXTSTATE <= RELEASE;
    PKT_RDY <= '0';
  end case;
end process;


DEBUG_CMDPROC(0) <= '1' when STATE = COPY_ACK else '0';
DEBUG_CMDPROC(1) <= '1' when STATE = RELEASE else '0';
```

```
DEBUG_CMDPROC(2) <= '1' when STATE = LOCK_CHANGES else
'0';
DEBUG_CMDPROC(3) <= '1' when OUTMODE = DDM else '0';
DEBUG_CMDPROC(4) <= '1' when OUTMODE = NULLPKT else '0';
DEBUG_CMDPROC(5) <= '1' when OUTMODE = FORWARD else '0';


end Behavioral;
```

9.2.8 DDMTable.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity DDM2 is

--Data from control registers
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_WRITE_L: in std_logic;
  BLOCK_ENABLE: in std_logic;
  BLOCK_DONE: out std_logic;

  THIS_ADDR: in T_NODEADDR;
  NETTIME: in T_NETTIME;

  --interface from command processor
  BLOCK_PKT_RDY: out std_logic;
  BLOCK_PKT_CLR: in std_logic;

  BLOCK_PKT_ADDR_TO_OUT: out T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_OUT: out T_NODEADDR;
  BLOCK_PKT_CMD_OUT: out T_COMMAND;
  BLOCK_PKT_DATA_OUT: out T_DATA_PAYLOAD;

  BLOCK_PKT_ADDR_TO_IN: in T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_IN: in T_NODEADDR;
  BLOCK_PKT_CMD_IN: in T_COMMAND;
  BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;
  BLOCK_PKT_GOOD: in std_logic;

  --interface to DSP
  BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
  BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0 );
  BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );

  DEBUG_DDM: out std_logic_vector( 15 downto 0 )
);
attribute ram_style: string;
--Ensure use of LUT based RAM.  (Other type is too slow,
and causes mapping problems).
--attribute ram_style: string;
attribute ram_style of DDM2: entity is "distributed";

end DDM2;

architecture Behavioral of DDM2 is

type T_STATE is (idle, setaddr, setdata1, setdata2,
setdata3, done );
signal STATE, NEXTSTATE: T_STATE;

--index of table used in DPRAM implementation
signal TINDEX: std_logic_vector( 2 downto 0 );
signal TINDEX2: std_logic_vector( 2 downto 0 );
signal TINDEX3: std_logic_vector( 2 downto 0 );
--signal TINDEXx: std_logic_vector( 3 downto 0 );
signal TINDEX2x: std_logic_vector( 2 downto 0 );
signal TINDEX3x: std_logic_vector( 2 downto 0 );

signal BLOCK_PKT_RDYX: std_logic;

--index into packet
signal WOFFSET: std_logic_vector( 3 downto 0 );

--Number of rows in table
constant C_ROWMAX : integer := 7;

--type for array of to addresses
type T_TOADDR_ARRAY  is array ( 0 to C_ROWMAX ) of
T_NODEADDR;
```

```
type T_PAYLOAD_ARRAY is array ( 0 to C_ROWMAX ) of
std_logic_vector( 31 downto 0 );
type T_PAYLOAD_ARRAY_SM is array ( 0 to C_ROWMAX ) of
std_logic_vector( 15 downto 0 );

type T_PKT_PROC_STATE is ( WAIT_NEW_DATA, TEST_ENO,
TEST_ENO2, WAIT_PKT_CLR,WAIT_SPECIAL_PKT_SETUP,
WAIT_SPECIAL_PKT_CLR,  COUNT_PKT_INDEX );
signal PKT_PROC_STATE: T_PKT_PROC_STATE;

--array of data to synchronize with normal data manager
signal NODE_DATA_OUT_TOADDR_ARRAY: T_TOADDR_ARRAY;
signal NODE_DATA_OUT_TOADDR_ARRAY2: T_TOADDR_ARRAY;
signal NODE_DATA_OUT_TOADDR_ARRAY3: T_TOADDR_ARRAY;

--for reading to dsp
signal NODE_DATA_OUT_TOADDR_ARRAYd: std_logic_vector( 7
downto 0 );
signal NODE_DATA_OUT_PAYLOAD1_ARRAYd: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD2_ARRAYd: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD3_ARRAYd: std_logic_vector(
15 downto 0 );


signal NODE_DATA_OUT_TOADDR_ARRAYx: T_NODEADDR;
signal NODE_DATA_OUT_TOADDR_ARRAY2x: T_NODEADDR;
signal NODE_DATA_OUT_TOADDR_ARRAY3x: T_NODEADDR;

signal NODE_DATA_OUT_PAYLOAD1_ARRAY: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD2_ARRAY: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD3_ARRAY: T_PAYLOAD_ARRAY_SM;
signal ENO: std_logic_vector( 31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD1_ARRAYx: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD2_ARRAYx: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD3_ARRAYx: std_logic_vector(
15 downto 0 );

signal NODE_DATA_OUT_PAYLOAD1_ARRAY2: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD2_ARRAY2: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD3_ARRAY2:
T_PAYLOAD_ARRAY_SM;
signal ENO2: std_logic_vector( 15 downto 0 );
signal NODE_DATA_OUT_PAYLOAD1_ARRAY2x: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD2_ARRAY2x: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD3_ARRAY2x: std_logic_vector(
15 downto 0 );

signal NODE_DATA_OUT_PAYLOAD1_ARRAY3: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD2_ARRAY3: T_PAYLOAD_ARRAY;
signal NODE_DATA_OUT_PAYLOAD3_ARRAY3:
T_PAYLOAD_ARRAY_SM;
signal ENO3: std_logic_vector( 15 downto 0 );
signal NODE_DATA_OUT_PAYLOAD1_ARRAY3x: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD2_ARRAY3x: std_logic_vector(
31 downto 0 );
signal NODE_DATA_OUT_PAYLOAD3_ARRAY3x: std_logic_vector(
15 downto 0 );

--DSP Read Interface
signal DATA_IN_SPECIAL1_BUFFER: std_logic_vector( 31
downto 0 );
signal DATA_IN_SPECIAL2_BUFFER: std_logic_vector( 31
downto 0 );
signal DATA_IN_SPECIAL3_BUFFER: std_logic_vector( 31
downto 0 );
signal DATA_IN_SPECIAL4_BUFFER: std_logic_vector( 31
downto 0 );
signal DATA_IN_XFER: std_logic; --trigger transfer
signal DATA_IN_SPECIAL1_RDY: std_logic_vector( 31 downto
0 );
signal DATA_IN_SPECIAL2_RDY: std_logic_vector( 31 downto
0 );
signal DATA_IN_SPECIAL3_RDY: std_logic_vector( 31 downto
0 );
signal DATA_IN_SPECIAL4_RDY: std_logic_vector( 31 downto
0 );
```

```
signal BLOCK_OUT_MUX00, BLOCK_OUT_MUX10,
BLOCK_OUT_MUX01: std_logic_vector( 31 downto 0 );

signal NULL_PKT: std_logic; --1 indicates the current
input packet is a null packet



signal FIRE2: std_logic;
signal FIRE3: std_logic;


type T_ESRD is record
  SET: std_logic;
  RST: std_logic;
  LOCKR: std_logic;
  LOCKS: std_logic;
  VAL: std_logic;
end record;

signal NEW_DATA: T_ESRD;
signal NEW_DATA2: T_ESRD;

signal PKT_INDEX: integer range 0 to C_ROWMAX;
signal PKT_INDEXx: integer range 0 to C_ROWMAX;
signal PKT_INDEX_PLUS_1: integer range 0 to C_ROWMAX;
signal PKT_INDEX_SAT: std_logic;

constant PKT_INDEX_MAX : integer :=  C_ROWMAX; --
indicates how much data is valid in DPRAM3

--there are two ways to implement this block.  one is
with PKT_INDEX_MAX as a constant
--and the data position in DPRAM 3 follows its position
within DPRAM2.  The other way
--is to make the transfer from DPRAM2 to DPRAM3
consolidate all the packets into the first part
--of the list, in which case, PKT_INDEX_MAX would be
counting how long the list is.  For now,
--we will scan the entire list using the high frequency
clock, and so it is a constant instead of a signal.

--special packet transmitter
signal SPECIAL_PKT_BUFFER1_PREP: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER2_PREP: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER3_PREP: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER4_PREP: std_logic_vector( 31
downto 0 );

signal SPECIAL_PKT_BUFFER1_PEND: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER2_PEND: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER3_PEND: std_logic_vector( 31
downto 0 );
signal SPECIAL_PKT_BUFFER4_PEND: std_logic_vector( 31
downto 0 );
signal BLOCK_DATA_OUT_RXTABLE: std_logic_vector( 31
downto 0 );

signal NEW_DATA_SPECIAL: T_ESRD;
signal TRANSFER_SPECIAL_PKT: std_logic;

signal BLOCK_WRITE_ENABLE_NORMAL: std_logic;
signal BLOCK_WRITE_ENABLE_SPECIAL: std_logic;
signal BLOCK_WRITE_ENABLE: std_logic;
signal BLOCK_ENABLE_TABLE: std_logic;

signal DDM_TX_STATUS: std_logic_vector( 31 downto 0 );
signal DDM_FSM_SPECIAL: std_logic;
signal DDM_FSM_NORMAL: std_logic;
signal DDM_FSM_IDLE: std_logic;
signal DDM_FSM_STATUS: std_logic_vector( 3 downto 0 );

signal EN_XFER2: std_logic;
--Format of commands to this driver:
--  ADDR:  [0001][Table Index][WORD OFFSET] - outbound
data
signal DEBUG_DDMX: std_logic_vector( 15 downto 0 );
```

```
component ADDR_REG is
generic(
  ADDR_VAL: std_logic_vector( 3 downto 0 ) );
port(
       CLK: in std_logic;
       RESET_L: in std_logic;
       WR_EN: in std_logic;
       ADDR_IN: in std_logic_vector( 3 downto 0 );
       DATA_IN: in std_logic_vector( 31 downto 0 );
       DATA_OUT: out std_logic_vector( 31 downto 0 ) );

end component;

component DDMRXTABLE is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_WRITE_L: in std_logic;
  BLOCK_ENABLE: in std_logic;


  BLOCK_PKT_GOOD: in std_logic;
  BLOCK_PKT_ADDR_FROM_IN: in T_NODEADDR;
  BLOCK_PKT_ADDR_TO_IN: in T_NODEADDR;
  BLOCK_PKT_CMD_IN: in T_COMMAND;
  BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;
  NET_TIME: in T_NETTIME;

  --interface to DSP
  BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
  BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0 );
  BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 )
    );
end component;

--================================================
begin
BLOCK_ENABLE_TABLE <= BLOCK_ENABLE and
BLOCK_ADDR_IN(15);

--Receive interface
```

```
C_DDMRXTABLE: DDMRXTABLE port map(
  BLOCK_HSCLK => BLOCK_HSCLK,
  BLOCK_RESET_L => BLOCK_RESET_L,
  BLOCK_ENABLE => BLOCK_ENABLE_TABLE,
  BLOCK_WRITE_L => BLOCK_WRITE_L,

  BLOCK_PKT_GOOD => BLOCK_PKT_GOOD,
  BLOCK_PKT_ADDR_FROM_IN  => BLOCK_PKT_ADDR_FROM_IN,
  BLOCK_PKT_ADDR_TO_IN => BLOCK_PKT_ADDR_TO_IN,
  BLOCK_PKT_CMD_IN => BLOCK_PKT_CMD_IN,
  BLOCK_PKT_DATA_IN => BLOCK_PKT_DATA_IN,
  NET_TIME => NETTIME,
  BLOCK_DATA_IN => BLOCK_DATA_IN,
  BLOCK_DATA_OUT => BLOCK_DATA_OUT_RXTABLE,
  BLOCK_ADDR_IN => BLOCK_ADDR_IN
);


BLOCK_PKT_RDY <= BLOCK_PKT_RDYX;

process( BLOCK_WRITE_ENABLE, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     BLOCK_WRITE_ENABLE <= '0';
    else
     if( BLOCK_ENABLE = '1' and BLOCK_WRITE_L = '0' )
then
       BLOCK_WRITE_ENABLE <= '1';
     else
       BLOCK_WRITE_ENABLE <= '0';
     end if; -- when  BLOCK_ENABLE = '1' and
BLOCK_WRITE_L = '0' else '0';
    end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge ( BLOCK_HSCLK ) ) then
    TINDEX <= BLOCK_ADDR_IN( 10 downto 8);
    WOFFSET <= BLOCK_ADDR_IN(4 downto 1 );
```

```
   if( BLOCK_ENABLE = '1' and BLOCK_WRITE_L = '0' and
WOFFSET(3) = '0' and BLOCK_ADDR_IN(15) = '0') then
    BLOCK_WRITE_ENABLE_NORMAL <= '1';
   else
    BLOCK_WRITE_ENABLE_NORMAL <= '0';
   end if;

   if( BLOCK_ENABLE = '1' and BLOCK_WRITE_L = '0' and
WOFFSET(3) = '1' and BLOCK_ADDR_IN(15) = '0' ) then
    BLOCK_WRITE_ENABLE_SPECIAL <= '1';
   else
    BLOCK_WRITE_ENABLE_SPECIAL <= '0';
   end if;

  end if;
end process;


--=======================================================
--DSP Interface to Protocol using DPRAM1
--=======================================================
process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    --The first bit is a '0', addressing this case
    if(  BLOCK_WRITE_ENABLE_NORMAL = '1' ) then
     case WOFFSET is
       when "0100" =>
         --store address of this line
         NODE_DATA_OUT_TOADDR_ARRAY( CONV_INTEGER( TINDEX
) ) <= BLOCK_DATA_IN(7 downto 0 );
         FIRE2 <= '0';
       when "0001" =>
         NODE_DATA_OUT_PAYLOAD1_ARRAY( CONV_INTEGER(
TINDEX ) ) <= BLOCK_DATA_IN;
         FIRE2 <= '0';
       when "0010" =>
         NODE_DATA_OUT_PAYLOAD2_ARRAY( CONV_INTEGER(
TINDEX ) ) <= BLOCK_DATA_IN;
         FIRE2 <= '0';
       when "0011" =>
         NODE_DATA_OUT_PAYLOAD3_ARRAY( CONV_INTEGER(
TINDEX ) ) <= BLOCK_DATA_IN(15 downto 0);
         FIRE2 <= '0';
       when "0111" =>
         --ENO <= BLOCK_DATA_IN( 15 downto 0 );
         FIRE2 <= '1';
       when others => null;
     end case;
    else
     FIRE2 <= '0';
    end if; --else    --read

    TINDEX2x <= TINDEX2;

  -- case WOFFSET is
  --   when "0001" =>   BLOCK_DATA_OUT              <=
NODE_DATA_OUT_PAYLOAD1_ARRAY( CONV_INTEGER( TINDEX ) );
  --   when "0010" =>   BLOCK_DATA_OUT               <=
NODE_DATA_OUT_PAYLOAD2_ARRAY( CONV_INTEGER( TINDEX ) );
  --   when "0011" =>   BLOCK_DATA_OUT(15 downto 0 ) <=
NODE_DATA_OUT_PAYLOAD3_ARRAY( CONV_INTEGER( TINDEX ) );
  --   when others => null;
  -- end case;
  end if;
end process;

--=======================================================
--Special Packet Buffer interface
--=======================================================
Q1: ADDR_REG generic map( "1000" )
  port map( BLOCK_HSCLK, BLOCK_RESET_L,
BLOCK_WRITE_ENABLE, WOFFSET,
       BLOCK_DATA_IN, SPECIAL_PKT_BUFFER1_PREP );

Q2: ADDR_REG generic map( "1001" )
  port map( BLOCK_HSCLK, BLOCK_RESET_L,
BLOCK_WRITE_ENABLE, WOFFSET,
       BLOCK_DATA_IN, SPECIAL_PKT_BUFFER2_PREP );

Q3: ADDR_REG generic map( "1010" )
  port map( BLOCK_HSCLK, BLOCK_RESET_L,
BLOCK_WRITE_ENABLE, WOFFSET,
```

```
       BLOCK_DATA_IN, SPECIAL_PKT_BUFFER3_PREP );

Q4: ADDR_REG generic map( "1011")
  port map( BLOCK_HSCLK, BLOCK_RESET_L,
BLOCK_WRITE_ENABLE, WOFFSET,
       BLOCK_DATA_IN, SPECIAL_PKT_BUFFER4_PREP );


Q5: ADDR_REG generic map( "0111" )
  port map( BLOCK_HSCLK, BLOCK_RESET_L,
BLOCK_WRITE_ENABLE, WOFFSET,
       BLOCK_DATA_IN, ENO );


--=======================================================
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK )  ) then
    if( BLOCK_RESET_L = '0' ) then
     TRANSFER_SPECIAL_PKT <= '0';
    else
     if( BLOCK_WRITE_ENABLE_SPECIAL = '1' and WOFFSET =
"1111" ) then
       TRANSFER_SPECIAL_PKT <= '1';
     else
       TRANSFER_SPECIAL_PKT <= '0';
     end if;
    end if;
  end if;

  --Transfer special packet to working buffer
  if( rising_edge( BLOCK_HSCLK ) ) then

    if( BLOCK_RESET_L = '0' ) then
     SPECIAL_PKT_BUFFER1_PEND <= X"00000000";
     SPECIAL_PKT_BUFFER2_PEND <= X"00000000";
     SPECIAL_PKT_BUFFER3_PEND <= X"00000000";
     SPECIAL_PKT_BUFFER4_PEND <= X"00000000";
     NEW_DATA_SPECIAL.SET <= '0';

    else
     if( TRANSFER_SPECIAL_PKT = '1' ) then
       SPECIAL_PKT_BUFFER1_PEND <=
SPECIAL_PKT_BUFFER1_PREP;
       SPECIAL_PKT_BUFFER2_PEND <=
SPECIAL_PKT_BUFFER2_PREP;
       SPECIAL_PKT_BUFFER3_PEND <=
SPECIAL_PKT_BUFFER3_PREP;
       SPECIAL_PKT_BUFFER4_PEND <=
SPECIAL_PKT_BUFFER4_PREP;
       NEW_DATA_SPECIAL.SET <= '1';
     else
       NEW_DATA_SPECIAL.SET <= '0';
     end if;
    end if;
  end if;
end process;

--=======================================================
--Second tier DPRAM process (DPRAM2)
--=======================================================
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable EN_XFER: std_logic;

variable FIRST_STATE: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK) ) then
    if( BLOCK_RESET_L = '0' ) then
     TINDEX2 <= "000";
     EN_XFER := '0';
     ENO2 <= X"0000";
     NEW_DATA.SET <= '0';
     FIRST_STATE := '0';
    else
     --Initiate Data Transfer
     if( FIRE2 = '1' and EN_XFER = '0') then
       EN_XFER := '1';

     --all data transferred
     elsif( TINDEX2 >= C_ROWMAX ) then
       EN_XFER := '0';
     end if;

     if( EN_XFER = '1' ) then
```

```
      --notify as soon as process has started
      NEW_DATA.SET <= '1';

      --Dualport ram has two clock cycles (address and
data) in this configuration
      if( FIRST_STATE = '1' ) then
        FIRST_STATE := '0';
      else
        FIRST_STATE := '1';
        TINDEX2 <= TINDEX2 + 1;
      end if;

      ENO2 <= "00000000" & ENO(7 downto 0 );
    else
      TINDEX2 <= "000";
      NEW_DATA.SET <= '0';
      FIRST_STATE := '0';
    end if;
   end if;
  end if;

  --Transfer data from first tier to second tier dual
port ram to make it available for system to use
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( EN_XFER = '1' ) then
     NODE_DATA_OUT_TOADDR_ARRAY2(  CONV_INTEGER(TINDEX2
)) <= NODE_DATA_OUT_TOADDR_ARRAY(  CONV_INTEGER(TINDEX2
));
     NODE_DATA_OUT_PAYLOAD1_ARRAY2(CONV_INTEGER(TINDEX2
)) <= NODE_DATA_OUT_PAYLOAD1_ARRAYx; --
(CONV_INTEGER(TINDEX2 ));
     NODE_DATA_OUT_PAYLOAD2_ARRAY2(CONV_INTEGER(TINDEX2
)) <= NODE_DATA_OUT_PAYLOAD2_ARRAYx; --
(CONV_INTEGER(TINDEX2 ));
     NODE_DATA_OUT_PAYLOAD3_ARRAY2(CONV_INTEGER(TINDEX2
)) <= NODE_DATA_OUT_PAYLOAD3_ARRAYx; --
(CONV_INTEGER(TINDEX2 ));
    end if;

    TINDEX2x <= TINDEX2;

  end if;
end process;
--signal NODE_DATA_OUT_PAYLOAD1_ARRAYx:
std_logic_vector( 31 downto 0 );
--signal NODE_DATA_OUT_PAYLOAD2_ARRAYx:
std_logic_vector( 31 downto 0 );
--signal NODE_DATA_OUT_PAYLOAD3_ARRAYx:
std_logic_vector( 15 downto 0 );

NODE_DATA_OUT_TOADDR_ARRAYx <=
NODE_DATA_OUT_TOADDR_ARRAY( CONV_INTEGER(TINDEX2x));
NODE_DATA_OUT_PAYLOAD1_ARRAYx <=
NODE_DATA_OUT_PAYLOAD1_ARRAY(CONV_INTEGER(TINDEX2x));
NODE_DATA_OUT_PAYLOAD2_ARRAYx <=
NODE_DATA_OUT_PAYLOAD2_ARRAY(CONV_INTEGER(TINDEX2x));
NODE_DATA_OUT_PAYLOAD3_ARRAYx <=
NODE_DATA_OUT_PAYLOAD3_ARRAY(CONV_INTEGER(TINDEX2x));


NODE_DATA_OUT_TOADDR_ARRAYd   <=
NODE_DATA_OUT_TOADDR_ARRAY(  CONV_INTEGER(TINDEX));
NODE_DATA_OUT_PAYLOAD1_ARRAYd <=
NODE_DATA_OUT_PAYLOAD1_ARRAY(CONV_INTEGER(TINDEX));
NODE_DATA_OUT_PAYLOAD2_ARRAYd <=
NODE_DATA_OUT_PAYLOAD2_ARRAY(CONV_INTEGER(TINDEX));
NODE_DATA_OUT_PAYLOAD3_ARRAYd <=
NODE_DATA_OUT_PAYLOAD3_ARRAY(CONV_INTEGER(TINDEX));


--------------------------------------------------------
--Third tier DPRAM implementation ("DPRAM3")
--------------------------------------------------------
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable FIRST_STATE: std_logic;
begin
-- if( BLOCK_RESET_L = '0' ) then
--   TINDEX3 <= "0000";
--   EN_XFER2 <= '0';
--   ENO3 <= X"0000";
--   NEW_DATA2.SET <= '0';
--   FIRST_STATE := '0';
  if( rising_edge( BLOCK_HSCLK) ) then
  if( BLOCK_RESET_L = '0' ) then
```

```
   TINDEX3 <= "000";
   EN_XFER2 <= '0';
   ENO3 <= X"0000";
   NEW_DATA2.SET <= '0';
   FIRST_STATE := '0';

  else
   --TINDEX3x <= TINDEX3;

   --if told to do so, start transferring data
   if( FIRE3 = '1' and EN_XFER2 = '0') then
    EN_XFER2 <= '1';
    NEW_DATA2.SET <= '0';
   --Data has finished transferring
   elsif( TINDEX3 >= C_ROWMAX ) then
    EN_XFER2 <= '0';
--     TINDEX3 <= "000";

    --notify after transfer has fully completed
    NEW_DATA2.SET <= '1';
   end if;

   --data is being transferred
   if( EN_XFER2 = '1' ) then
    --Dualport ram has two clock cycles (address and
data) in this configuration
    if( FIRST_STATE = '1' ) then
      FIRST_STATE := '0';
    else
      FIRST_STATE := '1';
      TINDEX3 <= TINDEX3 + 1;
    end if;

    ENO3 <= ENO2;
   else
    TINDEX3 <= "000";
   end if;
  end if;
  end if;
```

```
  --Transfer data from first tier to third tier dual
port ram to make it available for system to use
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( EN_XFER2 = '1' ) then
     NODE_DATA_OUT_TOADDR_ARRAY3(  CONV_INTEGER(TINDEX3
)) <= NODE_DATA_OUT_TOADDR_ARRAY2(  CONV_INTEGER(TINDEX3
));
     NODE_DATA_OUT_PAYLOAD1_ARRAY3(CONV_INTEGER(TINDEX3
)) <= NODE_DATA_OUT_PAYLOAD1_ARRAY2x; --
(CONV_INTEGER(TINDEX3 ));
     NODE_DATA_OUT_PAYLOAD2_ARRAY3(CONV_INTEGER(TINDEX3
)) <= NODE_DATA_OUT_PAYLOAD2_ARRAY2x; --
(CONV_INTEGER(TINDEX3 ));
     NODE_DATA_OUT_PAYLOAD3_ARRAY3(CONV_INTEGER(TINDEX3
)) <= NODE_DATA_OUT_PAYLOAD3_ARRAY2x; --
(CONV_INTEGER(TINDEX3 ));
    end if;

    TINDEX3x <= TINDEX3;
    PKT_INDEXx <= PKT_INDEX;
  end if;
end process;

NODE_DATA_OUT_TOADDR_ARRAY2x <=
NODE_DATA_OUT_TOADDR_ARRAY2( CONV_INTEGER(TINDEX3x));
NODE_DATA_OUT_PAYLOAD1_ARRAY2x <=
NODE_DATA_OUT_PAYLOAD1_ARRAY2(CONV_INTEGER(TINDEX3x));
NODE_DATA_OUT_PAYLOAD2_ARRAY2x <=
NODE_DATA_OUT_PAYLOAD2_ARRAY2(CONV_INTEGER(TINDEX3x));
NODE_DATA_OUT_PAYLOAD3_ARRAY2x <=
NODE_DATA_OUT_PAYLOAD3_ARRAY2(CONV_INTEGER(TINDEX3x));



--=======================================================
--Inter-process communication signals  (rising edge
triggered SET/RESET events)
--=======================================================
--Note: I had trouble making a procedure to do these, so
they are here as three instances:
-- NEW_DATA, NEW_DATA2, and NEW_DATA_SPECIAL
```

```
--NEW_DATA is set by DPRAM2 process
--NEW_DATA2 is set by DPRAM3
--Both are reset by the PKTDRIVER
--IPC signals used to synchronize transfers
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable LOCK1S, LOCK1R, LOCK2S, LOCK2R: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     NEW_DATA.VAL <= '0';
     NEW_DATA.LOCKS <= '0';
     NEW_DATA.LOCKR <= '0';

     NEW_DATA2.VAL <= '0';
     NEW_DATA2.LOCKS <= '0';
     NEW_DATA2.LOCKR <= '0';

     NEW_DATA_SPECIAL.VAL <= '0';
     NEW_DATA_SPECIAL.LOCKS <= '0';
     NEW_DATA_SPECIAL.LOCKR <= '0';

    else
     --Use a positive transition on SET/RESET signal for
NEW_DATA
     if( NEW_DATA.SET = '1' and NEW_DATA.LOCKS = '0')
then
       NEW_DATA.VAL <= '1';
       NEW_DATA.LOCKS <= '1';
     elsif( NEW_DATA.RST = '1' and NEW_DATA.LOCKR = '0' )
then
       NEW_DATA.VAL <= '0';
       NEW_DATA.LOCKR <= '1';
     end if;

     --reset lockout if signal falls
     if( NEW_DATA.RST = '0' ) then
       NEW_DATA.LOCKR <= '0';
     end if;

     --reset lockout if signal falls
     if( NEW_DATA.SET = '0' ) then
       NEW_DATA.LOCKS <= '0';
     end if;


     --Use SET/RESET signal for NEW_DATA
     if( NEW_DATA2.SET = '1' and NEW_DATA2.LOCKS = '0')
then
       NEW_DATA2.VAL <= '1';
       NEW_DATA2.LOCKS <= '1';
     elsif( NEW_DATA2.RST = '1' and NEW_DATA2.LOCKR = '0'
) then
       NEW_DATA2.VAL <= '0';
       NEW_DATA2.LOCKR <= '1';
     end if;

     --reset lockout if signal falls
     if( NEW_DATA2.RST = '0' ) then
       NEW_DATA2.LOCKR <= '0';
     end if;

     --reset lockout if signal falls
     if( NEW_DATA2.SET = '0' ) then
       NEW_DATA2.LOCKS <= '0';
     end if;

     --Special Packet buffer
     --Use SET/RESET signal for NEW_DATA
     if( NEW_DATA_SPECIAL.SET = '1' and
NEW_DATA_SPECIAL.LOCKS = '0') then
       NEW_DATA_SPECIAL.VAL <= '1';
       NEW_DATA_SPECIAL.LOCKS <= '1';
     elsif( NEW_DATA_SPECIAL.RST = '1' and
NEW_DATA_SPECIAL.LOCKR = '0' ) then
       NEW_DATA_SPECIAL.VAL <= '0';
       NEW_DATA_SPECIAL.LOCKR <= '1';
     end if;

     --reset lockout if signal falls
     if( NEW_DATA_SPECIAL.RST = '0' ) then
       NEW_DATA_SPECIAL.LOCKR <= '0';
     end if;

     --reset lockout if signal falls
```

```
    if( NEW_DATA_SPECIAL.SET = '0' ) then
      NEW_DATA_SPECIAL.LOCKS <= '0';
    end if;
   end if;
  end if;
end process;

--=======================================================
-- Command processor interface
--=======================================================
--Write from DPRAM3 to cmd processor
process( BLOCK_HSCLK, BLOCK_RESET_L ) is

begin
--pkt_index controller needed to reduce time from state
to packet
  if( rising_edge( BLOCK_HSCLK ) ) then
  if( BLOCK_RESET_L = '0' ) then
    PKT_PROC_STATE <= WAIT_NEW_DATA;
    NEW_DATA2.RST <= '0';
    NEW_DATA.RST <= '0';
    NEW_DATA_SPECIAL.RST <= '0';
    FIRE3 <= '0';

  else
    case PKT_PROC_STATE is
     --Wait for a new set of data to arrive
     when WAIT_NEW_DATA =>
       --if there is data ready, transfer it to working
buffer (DPRAM3)
       if( NEW_DATA.VAL = '1' ) then
         FIRE3 <= '1';  --initiate table update for third
table
         NEW_DATA.RST <= '1';  --acknowledge newdata 2
came and transaction processed
       else
         FIRE3 <= '0';
         NEW_DATA.RST <= '0';
       end if;

       --Start at index 0, and data is not ready
  --     PKT_INDEX <= 0;
         BLOCK_PKT_RDYX <= '0';

         --check to see if data transfer has completed
         if( NEW_DATA_SPECIAL.VAL = '1' ) then
           PKT_PROC_STATE <= WAIT_SPECIAL_PKT_SETUP;
           NEW_DATA_SPECIAL.RST <= '1';
         elsif( NEW_DATA2.VAL = '1' ) then
           --if so, move on and start testing
           PKT_PROC_STATE <= TEST_ENO;
           NEW_DATA2.RST <= '1';
           NEW_DATA_SPECIAL.RST <= '0'; --if not new data
special, then clear the reset signal
         else
           --otherwise keep waiting
           PKT_PROC_STATE <= WAIT_NEW_DATA;
           NEW_DATA2.RST <= '0';
           NEW_DATA_SPECIAL.RST <= '0';  --if not new data
special, then clear the reset signal

         end if;

      --put special packet out
      --and wait for clr signal to be zero
      when WAIT_SPECIAL_PKT_SETUP =>
         --Set data here, but it will not mean anything
until rdy is asserted next cycle
         BLOCK_PKT_CMD_OUT          <=
SPECIAL_PKT_BUFFER1_PEND(31 downto 28);
         BLOCK_PKT_ADDR_TO_OUT      <=
SPECIAL_PKT_BUFFER1_PEND(23 downto 16);
         BLOCK_PKT_ADDR_FROM_OUT    <=
SPECIAL_PKT_BUFFER1_PEND(15 downto  8);

         BLOCK_PKT_DATA_OUT( 3 ) <=
SPECIAL_PKT_BUFFER2_PEND( 7 downto  0); --
(CONV_INTEGER(PKT_INDEX ))( 7 downto  0);
         BLOCK_PKT_DATA_OUT( 2 ) <=
SPECIAL_PKT_BUFFER2_PEND(15 downto  8); --
(CONV_INTEGER(PKT_INDEX ))(15 downto  8);
         BLOCK_PKT_DATA_OUT( 1 ) <=
SPECIAL_PKT_BUFFER2_PEND(23 downto 16); --
(CONV_INTEGER(PKT_INDEX ))(23 downto 16);
```

```
      BLOCK_PKT_DATA_OUT( 0 ) <=
SPECIAL_PKT_BUFFER2_PEND(31 downto 24); --
(CONV_INTEGER(PKT_INDEX ))(31 downto 24);

      BLOCK_PKT_DATA_OUT( 7 ) <=
SPECIAL_PKT_BUFFER3_PEND( 7 downto  0); --
(CONV_INTEGER(PKT_INDEX ))( 7 downto  0);
      BLOCK_PKT_DATA_OUT( 6 ) <=
SPECIAL_PKT_BUFFER3_PEND(15 downto  8); --
(CONV_INTEGER(PKT_INDEX ))(15 downto  8);
      BLOCK_PKT_DATA_OUT( 5 ) <=
SPECIAL_PKT_BUFFER3_PEND(23 downto 16); --
(CONV_INTEGER(PKT_INDEX ))(23 downto 16);
      BLOCK_PKT_DATA_OUT( 4 ) <=
SPECIAL_PKT_BUFFER3_PEND(31 downto 24); --
(CONV_INTEGER(PKT_INDEX ))(31 downto 24);

      BLOCK_PKT_DATA_OUT( 9 ) <=
SPECIAL_PKT_BUFFER4_PEND( 7 downto  0); --
(CONV_INTEGER(PKT_INDEX ))( 7 downto  0);
      BLOCK_PKT_DATA_OUT( 8 ) <=
SPECIAL_PKT_BUFFER4_PEND(15 downto  8); --
(CONV_INTEGER(PKT_INDEX ))(15 downto  8);

      if( BLOCK_PKT_CLR = '0' ) then
        PKT_PROC_STATE <= WAIT_SPECIAL_PKT_CLR;
      else
        PKT_PROC_STATE <= WAIT_SPECIAL_PKT_SETUP;
      end if;

      BLOCK_PKT_RDYX <= '0';
      NEW_DATA_SPECIAL.RST <= '1';


    --wait for clr signal to go to 1, indicating packet
has been taken
    when WAIT_SPECIAL_PKT_CLR =>
      if( BLOCK_PKT_CLR = '1' ) then
        PKT_PROC_STATE <= WAIT_NEW_DATA;
      else
        PKT_PROC_STATE <= WAIT_SPECIAL_PKT_CLR;
      end if;

        BLOCK_PKT_RDYX <= '1';



    --See if the current packet is supposed to be sent
    --If so, send it, otherwise increment the counter
    --First ensure that clr is low before placing a new
packet on the bus
    when TEST_ENO =>
      PKT_PROC_STATE <= TEST_ENO2;
      FIRE3 <= '0';
    when TEST_ENO2 =>
      FIRE3 <= '0';  --reset update request (finished,
and ready for next one)

      if( ENO3(PKT_INDEX ) = '1' and BLOCK_PKT_CLR =
'0' ) then
        PKT_PROC_STATE <= WAIT_PKT_CLR;  -- packet is
valid and ready to send
      elsif( ENO3(PKT_INDEX) = '0' ) then
        PKT_PROC_STATE <= COUNT_PKT_INDEX;  --index
indicates no packet ready; move to next
      else
        PKT_PROC_STATE <= TEST_ENO;  --waiting for CLR
to become 0
      end if;

      BLOCK_PKT_RDYX <= '0';

      --Set data here, but it will not mean anything
until rdy is asserted next cycle
      BLOCK_PKT_CMD_OUT <= C_CMD_NORMAL;
      BLOCK_PKT_ADDR_TO_OUT <=
NODE_DATA_OUT_TOADDR_ARRAY3x; --(PKT_INDEX);
      BLOCK_PKT_ADDR_FROM_OUT <= THIS_ADDR;

      BLOCK_PKT_DATA_OUT( 3 ) <=
NODE_DATA_OUT_PAYLOAD1_ARRAY3x( 7 downto  0);
      BLOCK_PKT_DATA_OUT( 2 ) <=
NODE_DATA_OUT_PAYLOAD1_ARRAY3x(15 downto  8);
```

```
       BLOCK_PKT_DATA_OUT( 1 ) <=
NODE_DATA_OUT_PAYLOAD1_ARRAY3x(23 downto 16);
       BLOCK_PKT_DATA_OUT( 0 ) <=
NODE_DATA_OUT_PAYLOAD1_ARRAY3x(31 downto 24);

       BLOCK_PKT_DATA_OUT( 7 ) <=
NODE_DATA_OUT_PAYLOAD2_ARRAY3x( 7 downto  0);
       BLOCK_PKT_DATA_OUT( 6 ) <=
NODE_DATA_OUT_PAYLOAD2_ARRAY3x(15 downto  8);
       BLOCK_PKT_DATA_OUT( 5 ) <=
NODE_DATA_OUT_PAYLOAD2_ARRAY3x(23 downto 16);
       BLOCK_PKT_DATA_OUT( 4 ) <=
NODE_DATA_OUT_PAYLOAD2_ARRAY3x(31 downto 24);

       BLOCK_PKT_DATA_OUT( 9 ) <=
NODE_DATA_OUT_PAYLOAD3_ARRAY3x( 7 downto  0);
       BLOCK_PKT_DATA_OUT( 8 ) <=
NODE_DATA_OUT_PAYLOAD3_ARRAY3x(15 downto  8);


    --Packet is valid, and has been sent (BLOCK_PKT_CLR
= '1').
    when WAIT_PKT_CLR =>
      BLOCK_PKT_RDYX <= '1';

      if( BLOCK_PKT_CLR = '0' ) then
        PKT_PROC_STATE <= WAIT_PKT_CLR;
      else
        PKT_PROC_STATE <= COUNT_PKT_INDEX;
      end if;

    --only stay here one clock cycle.  Otherwise, will
count twice on PKT_INDEX
    when COUNT_PKT_INDEX =>
      BLOCK_PKT_RDYX <= '0';

  --     PKT_INDEX <= PKT_INDEX_PLUS_1; --PKT_INDEX + 1;

      if( PKT_INDEX_SAT = '0' ) then
        PKT_PROC_STATE <= TEST_ENO;

      else
        PKT_PROC_STATE <= WAIT_NEW_DATA;
       end if;
    end case;
  end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
variable SAT: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
  if( BLOCK_RESET_L = '0' ) then
    SAT := '0';
    PKT_INDEX_PLUS_1 <= 0;
  else
    if( PKT_INDEX < C_ROWMAX ) then
     SAT := '0';
    else
     SAT := '1';
    end if;

    if( SAT = '0' ) then
     PKT_INDEX_PLUS_1 <= PKT_INDEX + 1;
    else
     PKT_INDEX_PLUS_1 <= PKT_INDEX;
    end if;
  end if;
  end if;
end process;

--Part of the Dualport ram interface secondary side
NODE_DATA_OUT_TOADDR_ARRAY3x       <=
NODE_DATA_OUT_TOADDR_ARRAY3(CONV_INTEGER(PKT_INDEX));
NODE_DATA_OUT_PAYLOAD1_ARRAY3x <=
NODE_DATA_OUT_PAYLOAD1_ARRAY3(CONV_INTEGER(PKT_INDEX));
NODE_DATA_OUT_PAYLOAD2_ARRAY3x <=
NODE_DATA_OUT_PAYLOAD2_ARRAY3(CONV_INTEGER(PKT_INDEX));
NODE_DATA_OUT_PAYLOAD3_ARRAY3x <=
NODE_DATA_OUT_PAYLOAD3_ARRAY3(CONV_INTEGER(PKT_INDEX));

--test for saturation of packet index
process( BLOCK_HSCLK )is
```

```
variable RST_PKT_CTR: std_logic;
variable INCR_PKT_CTR: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
  if( BLOCK_RESET_L = '0' ) then
    PKT_INDEX_SAT <= '0';
    PKT_INDEX <= 0;
    RST_PKT_CTR := '1';
  else
    if( PKT_INDEX < PKT_INDEX_MAX ) then
     PKT_INDEX_SAT <= '0';
    else
     PKT_INDEX_SAT <= '1';
    end if;

    if( INCR_PKT_CTR = '1' ) then
     PKT_INDEX <= PKT_INDEX_PLUS_1;
    elsif( RST_PKT_CTR = '1' ) then
     PKT_INDEX <= 0;
    end if;

    if( PKT_PROC_STATE = WAIT_NEW_DATA ) then
     RST_PKT_CTR := '1';
    else
     RST_PKT_CTR := '0';
    end if;

    if( PKT_PROC_STATE = COUNT_PKT_INDEX ) then
     INCR_PKT_CTR := '1';
    else
     INCR_PKT_CTR := '0';
    end if;
  end if;
  end if;
end process;

DEBUG_DDMX(15) <= FIRE3;
DEBUG_DDMX(14) <= FIRE2;
DEBUG_DDMX(13) <= NEW_DATA.VAL;
DEBUG_DDMX(12) <= ENO(1);
DEBUG_DDMX(11)  <= '1' when PKT_PROC_STATE =
WAIT_NEW_DATA else '0';
DEBUG_DDMX(10)  <= '1' when PKT_PROC_STATE = TEST_ENO
else '0';
DEBUG_DDMX(9)  <= '1' when PKT_PROC_STATE = TEST_ENO2
else '0';
DEBUG_DDMX(8)  <= '1' when PKT_PROC_STATE = WAIT_PKT_CLR
else '0';
DEBUG_DDMX(7 downto 4) <= ENO2(3 downto 0);
DEBUG_DDMX(3 downto 0) <= ENO3(3 downto 0);

process( BLOCK_RESET_L, BLOCK_HSCLK )
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     DEBUG_DDM <= X"0000";
    else
     DEBUG_DDM <= DEBUG_DDMX;
    end if;
  end if;
end process;


P0: process( BLOCK_RESET_L, BLOCK_ENABLE, BLOCK_HSCLK )
begin
-- if( BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0') then
--   STATE <= idle;
  if( rising_edge(BLOCK_HSCLK) ) then
    if( BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXTSTATE;
    end if;
  end if;
end process;

--state machine for managing DSP writes
process( STATE )
begin
  case ( STATE ) is
    when idle =>
     NEXTSTATE <= setaddr;
     BLOCK_DONE <= '1';
```

```
   --for dual port ram, need to consider the address
setup time
   --because we are using synchronous ram
      when setaddr =>
    NEXTSTATE <= setdata1;
    BLOCK_DONE <= '0';
   --let the data set
   when setdata1 =>
    NEXTSTATE <= setdata2;
    BLOCK_DONE <= '0';
   --indicate done with transaction
   when setdata2 =>
    NEXTSTATE <= setdata3;
    BLOCK_DONE <= '0';
   when setdata3 =>
    NEXTSTATE <= done;
    BLOCK_DONE <= '0';
   when done =>
    NEXTSTATE <= done;
    BLOCK_DONE <= '1';
  end case;
end process;


--=======================================================
--DSP Read Interface

process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  --if( BLOCK_RESET_L = '0' ) then
  -- NULL_PKT <= '0';
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     NULL_PKT <= '0';
    else
     if( BLOCK_PKT_CMD_IN = C_CMD_NULL ) then
       NULL_PKT <= '1';
     else
       NULL_PKT <= '0';
     end if;
    end if;
  end if;
end process;

process( BLOCK_RESET_L, BLOCK_HSCLK ) is
variable LOCKOUT: std_logic;
begin
  if( rising_Edge( BLOCK_HSCLK )  ) then
    if( BLOCK_RESET_L = '0' ) then
     DATA_IN_SPECIAL1_BUFFER      <= X"00000000";
     DATA_IN_SPECIAL2_BUFFER      <= X"00000000";
     DATA_IN_SPECIAL3_BUFFER      <= X"00000000";
     DATA_IN_SPECIAL4_BUFFER      <= X"00000000";

     DATA_IN_SPECIAL1_RDY  <= X"00000000";
     DATA_IN_SPECIAL2_RDY  <= X"00000000";
     DATA_IN_SPECIAL3_RDY  <= X"00000000";
     DATA_IN_SPECIAL4_RDY  <= X"00000000";
     LOCKOUT := '0';

    else
     if( DATA_IN_XFER = '1' ) then
       DATA_IN_SPECIAL1_RDY <= DATA_IN_SPECIAL1_BUFFER;
       DATA_IN_SPECIAL2_RDY <= DATA_IN_SPECIAL2_BUFFER;
       DATA_IN_SPECIAL3_RDY <= DATA_IN_SPECIAL3_BUFFER;
       DATA_IN_SPECIAL4_RDY <= DATA_IN_SPECIAL4_BUFFER;
     end if;

     if( LOCKOUT = '0' and BLOCK_PKT_GOOD = '1' and
NULL_PKT = '0' ) then
       DATA_IN_SPECIAL1_BUFFER <= BLOCK_PKT_CMD_IN &
X"0" & BLOCK_PKT_ADDR_TO_IN & BLOCK_PKT_ADDR_FROM_IN &
NETTIME;
       DATA_IN_SPECIAL2_BUFFER <= BLOCK_PKT_DATA_IN(0) &
BLOCK_PKT_DATA_IN(1) & BLOCK_PKT_DATA_IN(2) &
BLOCK_PKT_DATA_IN(3);
       DATA_IN_SPECIAL3_BUFFER <= BLOCK_PKT_DATA_IN(4) &
BLOCK_PKT_DATA_IN(5) & BLOCK_PKT_DATA_IN(6) &
BLOCK_PKT_DATA_IN(7);
       DATA_IN_SPECIAL3_BUFFER <= BLOCK_PKT_DATA_IN(8) &
BLOCK_PKT_DATA_IN(9) & X"0000";
       LOCKOUT := '1';
     elsif( BLOCK_PKT_GOOD = '0' ) then
       LOCKOUT := '0';
```

```
    end if;
   end if;
  end if;
end process;

process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
-- if( BLOCK_RESET_L = '0' ) then
--  DATA_IN_XFER <= '0';
  if( rising_edge( BLOCK_HSCLK ) ) then
   if( BLOCK_RESET_L = '0' ) then
    DATA_IN_XFER <= '0';
   else
    if( BLOCK_ADDR_IN(4 downto 1) = "1100" and
BLOCK_WRITE_L = '0' and BLOCK_ENABLE = '1' ) then
      DATA_IN_XFER <= '1';
    else
      DATA_IN_XFER <= '0';
    end if;
   end if;
  end if;
end process;

--Manage DSP reads
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
   if( BLOCK_RESET_L = '0' ) then
    BLOCK_DATA_OUT <= X"00000000";
   else
    if( BLOCK_ADDR_IN(15) = '0' ) then
      case BLOCK_ADDR_IN(4 downto 3 )  is
      when "00"  => BLOCK_DATA_OUT <= BLOCK_OUT_MUX00;
      when "01"  => BLOCK_DATA_OUT <= BLOCK_OUT_MUX01;
      when "10"  => BLOCK_DATA_OUT <= BLOCK_OUT_MUX10;
      when others =>   BLOCK_DATA_OUT <= X"7E7E7E7E";  -
-error code for invalid address
      end case;
    else
      BLOCK_DATA_OUT <= BLOCK_DATA_OUT_RXTABLE;
    end if;
```

```
    case BLOCK_ADDR_IN(2 downto 1 )  is
    when "00" => BLOCK_OUT_MUX00 <= X"5a0000" &
NODE_DATA_OUT_TOADDR_ARRAYd;
    when "01" => BLOCK_OUT_MUX00 <=
NODE_DATA_OUT_PAYLOAD1_ARRAYd;
    when "10" => BLOCK_OUT_MUX00<=
NODE_DATA_OUT_PAYLOAD2_ARRAYd;
    when "11" => BLOCK_OUT_MUX00 <= X"a500" &
NODE_DATA_OUT_PAYLOAD3_ARRAYd;
    when others => BLOCK_OUT_MUX00 <= X"7E7E7E7E";
    end case;

    case BLOCK_ADDR_IN(2 downto 1 )  is
    when "00" => BLOCK_OUT_MUX01 <= X"5a0000" &
NODE_DATA_OUT_TOADDR_ARRAYd;
--     when "01" => BLOCK_OUT_MUX01 <= X"7E7E7E7E";
    when "10" => BLOCK_OUT_MUX01 <= DDM_TX_STATUS;
    when "11" => BLOCK_OUT_MUX01 <= ENO;
    when others => BLOCK_OUT_MUX01 <= X"7E7E7E7E";
    end case;

    case BLOCK_ADDR_IN(2 downto 1 )  is
    when "00" => BLOCK_OUT_MUX10 <=
DATA_IN_SPECIAL1_RDY;
    when "01" => BLOCK_OUT_MUX10 <=
DATA_IN_SPECIAL2_RDY;
    when "10" => BLOCK_OUT_MUX10 <=
DATA_IN_SPECIAL3_RDY;
    when "11" => BLOCK_OUT_MUX10 <=
DATA_IN_SPECIAL4_RDY;
    when others => BLOCK_OUT_MUX10 <= X"7E7E7E7E";
    end case;

    if( PKT_PROC_STATE = WAIT_SPECIAL_PKT_SETUP or
      PKT_PROC_STATE = WAIT_SPECIAL_PKT_CLR ) then
      DDM_FSM_SPECIAL <= '1';
    else
      DDM_FSM_SPECIAL <= '0';
    end if;

    if( PKT_PROC_STATE = TEST_ENO or
    PKT_PROC_STATE = TEST_ENO2 or
```

```
    PKT_PROC_STATE = WAIT_PKT_CLR or
    PKT_PROC_STATE = COUNT_PKT_INDEX ) then
      DDM_FSM_NORMAL <= '1';
    else
      DDM_FSM_NORMAL <= '0';
    end if;

    if( PKT_PROC_STATE = WAIT_NEW_DATA ) then
      DDM_FSM_IDLE <= '1';
    else
      DDM_FSM_IDLE <= '0';
    end if;

    DDM_TX_STATUS <= X"00" & DDM_FSM_STATUS &
CONV_STD_LOGIC_VECTOR( PKT_INDEX, 4) & NETTIME &
THIS_ADDR;
   end if;
  end if;
end process;

DDM_FSM_STATUS <= "0" & DDM_FSM_SPECIAL & DDM_FSM_NORMAL
& DDM_FSM_IDLE;

end Behavioral;
```

9.2.9 Addr_Reg.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ADDR_REG is
generic(
  ADDR_VAL: std_logic_vector( 3 downto 0 ) );
port(
       CLK: in std_logic;
       RESET_L: in std_logic;
       WR_EN: in std_logic;
       ADDR_IN: in std_logic_vector( 3 downto 0 );
--       ADDR_VAL: in std_logic_vector( 3 downto 0 );
       DATA_IN: in std_logic_vector( 31 downto 0 );
       DATA_OUT: out std_logic_vector( 31 downto 0 ) );

end ADDR_REG;

architecture Behavioral of ADDR_REG is
signal ADDR_MATCH: std_logic;
begin

process( RESET_L, CLK ) is
begin
    if( rising_edge( CLK ) ) then
     if( RESET_L = '0' ) then
       DATA_OUT <= CONV_STD_LOGIC_VECTOR( 0, 32);
     else
       if( ADDR_MATCH = '1' and WR_EN= '1' ) then
         DATA_OUT <= DATA_IN;
       end if;

       if( ADDR_IN = ADDR_VAL ) then
         ADDR_MATCH <= '1';
       else
         ADDR_MATCH <= '0';
         end if;
       end if;
     end if;
   end if;
end process;
end Behavioral;
```

9.2.10 DDMRxTable.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

--====================================
--Governs receied pakets from PESNet
entity DDMRXTABLE is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_WRITE_L: in std_logic;
  BLOCK_ENABLE: in std_logic;


  BLOCK_PKT_GOOD: in std_logic;
  BLOCK_PKT_ADDR_FROM_IN: in T_NODEADDR;
  BLOCK_PKT_ADDR_TO_IN: in T_NODEADDR;
  BLOCK_PKT_CMD_IN: in T_COMMAND;
  BLOCK_PKT_DATA_IN: in T_DATA_PAYLOAD;
  NET_TIME: in T_NETTIME;

  --interface to DSP
  BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
  BLOCK_ADDR_IN: in std_logic_vector( 15 downto 0 );
  BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 )
    );
end DDMRXTABLE;

architecture Behavioral of DDMRXTABLE is

--determines the number of positions for received data
constant C_ROWMAX: integer := 15;

component EVENT_VECTOR is
generic( NVEC: integer := 31 );
port(
  BLOCK_RESET_L: in std_logic;
  BLOCK_HSCLK: in std_logic;
  S: in std_logic_vector( NVEC - 1 downto 0 ); --set
(postiive edge triggered, synchronous)
  R: in std_logic_vector( NVEC - 1 downto 0 );  --reset
(positive edge triggered, synchronous)
  E: out std_logic_vector( NVEC - 1 downto 0 )  --event
out
);
end component;
  component PIPE_COMPARE is
  generic(
    VECSIZE: integer; RST_VAL: std_logic := '0');
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;
    VALUE: in std_logic_vector( VECSIZE - 1 downto 0 );
    TARGET: in std_logic_vector( VECSIZE - 1 downto 0 );
    RESULT: out std_logic);

  end component;

component ddm_rx_fifo_dp is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  CMD_IN: in T_COMMAND;
  ADDR_TO_IN: in T_NODEADDR;
  ADDR_FROM_IN: in T_NODEADDR;
  DATA_IN: in T_DATA_PAYLOAD;
  NETTIME_IN: in T_NETTIME;

  CMD_OUT: out T_COMMAND;
  ADDR_TO_OUT: out T_NODEADDR;
  ADDR_FROM_OUT: out T_NODEADDR;
```

```
  DATA_OUT: out T_DATA_PAYLOAD;
  NETTIME_OUT: out T_NETTIME;

  SIZE: out std_logic_vector( 3 downto 0 );

  R: in std_logic;
  W: in std_logic;

  EMPTY: out std_logic;
  FULL: out std_logic
);
end component;


signal NEW_DATA: std_logic_vector( C_ROWMAX downto 0 );

--type for array of to addresses
type T_BIG_ARRAY is array ( 0 to C_ROWMAX ) of
std_logic_vector( 31 downto 0 );
type T_ADDR_ARRAY is array ( 0 to C_ROWMAX ) of
T_NODEADDR;

--DPRAM interface
signal ADDR_LATCH: std_logic_vector( 15 downto 0 );

--DSP interface
signal RXDATA_ARRAY1: T_BIG_ARRAY;  -- contents:  [ D0
| D1      | D2 | D3 ]
signal RXDATA_ARRAY2: T_BIG_ARRAY;       -- contents:  [
D4      | D5      | D6 | D7 ]
signal RXDATA_ARRAY3: T_BIG_ARRAY;       -- contents:  [
TO_ADDR | NETTIME | D8 | D9 ]
signal FROM_ADDR_ARRAY: T_ADDR_ARRAY;  --addres to use
as lookup

signal LOOKUP_DATA: T_NODEADDR;
--signal LOOKUP_ADDR: std_logic_vector( 3 downto 0);
signal LOOKUP_MATCH_VECTOR: std_logic_vector( C_ROWMAX
downto 0);
signal LOOKUP_MATCH: std_logic;
--signal LOOKUP_ENABLE: std_logic;

signal BLOCK_DATA_OUTA: std_logic_vector( 7 downto 0 );
signal BLOCK_DATA_OUT1: std_logic_vector( 31 downto 0 );
signal BLOCK_DATA_OUT2: std_logic_vector( 31 downto 0 );
signal BLOCK_DATA_OUT3: std_logic_vector( 31 downto 0 );

signal WR_EN: std_logic_vector( 15 downto 0 );

signal ENC16: std_logic_vector(  3 downto 0 );

--used for pipelined encoding
signal ZONE: std_logic_vector( 3 downto 0 );

--encoded zones  (bitranges)
signal Z0R, Z1R, Z2R, Z3R: std_logic_vector( 3 downto 0
);

signal NEW_DATA_SET: std_logic_vector( C_ROWMAX downto 0
);
signal NEW_DATA_EVENT: std_logic_vector( C_ROWMAX downto
0 );
signal NEW_DATA_RESET: std_logic_vector( C_ROWMAX downto
0 );

signal FIFO_DATA_OUT1: std_logic_vector( 31 downto 0 );
signal FIFO_DATA_OUT2: std_logic_vector( 31 downto 0 );
signal FIFO_DATA_OUT3: std_logic_vector( 31 downto 0 );
signal FIFO_DATA_OUT4: std_logic_vector( 31 downto 0 );
signal FIFO_DATA_PAYLOAD: T_DATA_PAYLOAD;
signal FIFO_WRITE, FIFO_READ: std_logic;

signal CMD_NORMAL: std_logic;
signal CMD_NULL: std_logic;
begin

C_EVENT_VECTOR: EVENT_VECTOR generic map( NVEC =>
C_ROWMAX + 1 )
port map(
  BLOCK_HSCLK => BLOCK_HSCLK,
  BLOCK_RESET_L => BLOCK_RESET_L,
  S => NEW_DATA_SET,
  R => NEW_DATA_RESET,
  E => NEW_DATA_EVENT
```

```
  );

--=======================================================
--Pipelined encoder
--=======================================================
--Enocder implementation
--Uses pipelined decoder.  To implement entire encoder
in 1 clock cycle introduced
--too much delay, and so it is broken down into a
pipelined version.
--Zones are segments of the bit vector to encode.  Each
segment is 4 bits wide.
--Each zone is encoded by itself (they assume the rest
of the bit vector is zero
--as it should be). The results are placed in the
vectors Z3R, Z2R, Z1R, Z0R.
--If a ZONE is all zero, the corresponding element of
the vector ZONE is set to 0.
--Based on which bit of zone (only one should be
nonzero) is active, the result is taken
--from the corresponding previously encoded ZxR
register, and that is the final encoded value.
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( LOOKUP_MATCH_VECTOR( 15 downto 12) = "0000" )
then
     ZONE(3) <= '0';
    else
     ZONE(3) <= '1';
    end if;

    if( LOOKUP_MATCH_VECTOR( 11 downto 8) = "0000" ) then
     ZONE(2) <= '0';
    else
     ZONE(2) <= '1';
    end if;

    if( LOOKUP_MATCH_VECTOR( 7 downto 4) = "0000" ) then
     ZONE(1) <= '0';
    else
     ZONE(1) <= '1';
    end if;

    if( LOOKUP_MATCH_VECTOR( 3 downto 0) = "0000" ) then
     ZONE(0) <= '0';
    else
     ZONE(0) <= '1';
    end if;

    case LOOKUP_MATCH_VECTOR(15 downto 12) is
    when X"8" => Z3R <= X"F";
    when X"4" => Z3R <= X"E";
    when X"2" => Z3R <= X"D";
    when X"1" => Z3R <= X"C";
    when others => Z3R <= X"0";
    end case;

    case LOOKUP_MATCH_VECTOR(11 downto 8) is
    when X"8" => Z2R <= X"B";
    when X"4" => Z2R <= X"A";
    when X"2" => Z2R <= X"9";
    when X"1" => Z2R <= X"8";
    when others => Z2R <= X"0";
    end case;

    case LOOKUP_MATCH_VECTOR(7 downto 4) is
    when X"8" => Z1R <= X"7";
    when X"4" => Z1R <= X"6";
    when X"2" => Z1R <= X"5";
    when X"1" => Z1R <= X"4";
    when others => Z1R <= X"0";
    end case;

    case LOOKUP_MATCH_VECTOR(3 downto 0) is
    when X"8" => Z0R <= X"3";
    when X"4" => Z0R <= X"2";
    when X"2" => Z0R <= X"1";
    when X"1" => Z0R <= X"0";
    when others => Z0R <= X"0";
    end case;

    case ZONE is
    when X"8" => ENC16 <= Z3R;
```

```
          LOOKUP_MATCH <= '1';
   when X"4" => ENC16 <= Z2R;
          LOOKUP_MATCH <= '1';
   when X"2" => ENC16 <= Z1R;
          LOOKUP_MATCH <= '1';
   when X"1" => ENC16 <= Z0R;
          LOOKUP_MATCH <= '1';
   when others => ENC16 <= X"0";
            LOOKUP_MATCH <= '0';
   end case;

  end if;
end process;

--=======================================================
--Manage Address reads and writes
--=======================================================

--find the address of the memory location with data
containing the node address (context search)
LOOKUP_DATA <= BLOCK_PKT_ADDR_FROM_IN;

--implement address finder
G1: for i in 0 to C_ROWMAX generate
  process( BLOCK_RESET_L, BLOCK_HSCLK ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' ) then
       WR_EN(i) <= '0';
       FROM_ADDR_ARRAY(i) <= X"00";
     else
       --address matching for CAM like operation
       if( FROM_ADDR_ARRAY(i) = LOOKUP_DATA ) then
         LOOKUP_MATCH_VECTOR(i) <= '1';
       else
         LOOKUP_MATCH_VECTOR(i) <= '0';
       end if;

       --write enable
       if( BLOCK_WRITE_L = '0' and BLOCK_ADDR_IN(4
downto 1 ) = i and BLOCK_ADDR_IN(15) = '1' ) then
         WR_EN(i) <= '1';
       else
         WR_EN(i) <= '0';
       end if;

       if( WR_EN(i) = '1' and BLOCK_ENABLE = '1' ) then
         FROM_ADDR_ARRAY(i) <= BLOCK_DATA_IN(7 downto 0);
         NEW_DATA_SET(i) <= '1';
       end if;

     end if;
    end if;
  end process;
end generate;


--Process
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     BLOCK_DATA_OUTA <= X"00";
     BLOCK_DATA_OUT1 <= X"00000000";
     BLOCK_DATA_OUT2 <= X"00000000";
     BLOCK_DATA_OUT3 <= X"00000000";
    else
     ADDR_LATCH <= BLOCK_ADDR_IN( 15 downto 0 );
     BLOCK_DATA_OUTA <= FROM_ADDR_ARRAY(
CONV_INTEGER(ADDR_LATCH( 4 downto 1 )));
     BLOCK_DATA_OUT1 <=
RXDATA_ARRAY1(CONV_INTEGER(ADDR_LATCH(4 downto 1 )));
     BLOCK_DATA_OUT2 <=
RXDATA_ARRAY2(CONV_INTEGER(ADDR_LATCH(4 downto 1 )));
     BLOCK_DATA_OUT3 <=
RXDATA_ARRAY3(CONV_INTEGER(ADDR_LATCH(4 downto 1 )));

     --decode different pieces
     case ADDR_LATCH( 11 downto 8 ) is
     when X"0" =>
       BLOCK_DATA_OUT(7 downto 0 ) <= BLOCK_DATA_OUTA;
     when X"1" =>
       BLOCK_DATA_OUT <= BLOCK_DATA_OUT1;
     when X"2" =>
```

```
      BLOCK_DATA_OUT <= BLOCK_DATA_OUT2;
    when X"3" =>
      BLOCK_DATA_OUT <= BLOCK_DATA_OUT3;
    when X"4" =>
      BLOCK_DATA_OUT(15 downto 0 ) <= NEW_DATA;
      BLOCK_DATA_OUT(31 downto 16 ) <= X"0000";
    when X"5" =>
      BLOCK_DATA_OUT <= FIFO_DATA_OUT1;
    when X"6" =>
      BLOCK_DATA_OUT <= FIFO_DATA_OUT2;
    when X"7" =>
      BLOCK_DATA_OUT <= FIFO_DATA_OUT3;
    when X"8" =>
      BLOCK_DATA_OUT <= FIFO_DATA_OUT4;
    when others =>
      BLOCK_DATA_OUT <= X"DEDEDEDE";
    end case;
   end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
variable CMD_NORMALX: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( LOOKUP_MATCH = '1' and BLOCK_PKT_GOOD = '1' and
CMD_NORMALX = '1' ) then
--     RXDATA_ARRAY1(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_DATA_IN(3) & BLOCK_PKT_DATA_IN(2) &
BLOCK_PKT_DATA_IN(1) & BLOCK_PKT_DATA_IN(0);
--     RXDATA_ARRAY2(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_DATA_IN(7) & BLOCK_PKT_DATA_IN(6) &
BLOCK_PKT_DATA_IN(5) & BLOCK_PKT_DATA_IN(4);
--     RXDATA_ARRAY3(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_ADDR_TO_IN & NET_TIME & BLOCK_PKT_DATA_IN(9) &
BLOCK_PKT_DATA_IN(8);
     RXDATA_ARRAY1(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_DATA_IN(0) & BLOCK_PKT_DATA_IN(1) &
BLOCK_PKT_DATA_IN(2) & BLOCK_PKT_DATA_IN(3);
     RXDATA_ARRAY2(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_DATA_IN(4) & BLOCK_PKT_DATA_IN(5) &
BLOCK_PKT_DATA_IN(6) & BLOCK_PKT_DATA_IN(7);
     RXDATA_ARRAY3(CONV_INTEGER(ENC16)) <=
BLOCK_PKT_ADDR_TO_IN & NET_TIME & BLOCK_PKT_DATA_IN(8) &
BLOCK_PKT_DATA_IN(9);
    end if;

    --command is normal and it is a response containing
sensor information
    if( BLOCK_PKT_CMD_IN = C_CMD_NORMAL and
BLOCK_PKT_DATA_IN(1)(0) = '1') then
     CMD_NORMALX := '1';
    else
     CMD_NORMALX := '0';
    end if;
  end if;
end process;

PIPE_COMPARE1: PIPE_COMPARE generic map (VECSIZE => 4)
    port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_CMD_IN,
       TARGET => C_CMD_NORMAL, RESULT => CMD_NORMAL );

PIPE_COMPARE2: PIPE_COMPARE generic map (VECSIZE => 4)
    port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_CMD_IN,
       TARGET => C_CMD_NULL, RESULT => CMD_NULL );

------------------------------------------------------
----------------FIFO INTERFACE ----------------------
C_DDM_RX_FIFO_DP: DDM_RX_FIFO_DP port map(
  BLOCK_HSCLK => BLOCK_HSCLK,
  BLOCK_RESET_L => BLOCK_RESET_L,

  CMD_IN    => BLOCK_PKT_CMD_IN,
  ADDR_TO_IN  => BLOCK_PKT_ADDR_TO_IN,
  ADDR_FROM_IN    => BLOCK_PKT_ADDR_FROM_IN,
  DATA_IN   => BLOCK_PKT_DATA_IN,
  NETTIME_IN  => NET_TIME,
  W     => FIFO_WRITE,
  R     => FIFO_READ,
  SIZE     => FIFO_DATA_OUT1(31 downto 28 ),
  CMD_OUT   => FIFO_DATA_OUT1( 27 downto 24),
  ADDR_TO_OUT    => FIFO_DATA_OUT1(23 downto 16),
```

```
  ADDR_FROM_OUT  => FIFO_DATA_OUT1(15 downto 8),
  NETTIME_OUT => FIFO_DATA_OUT1(7 downto 0),
  DATA_OUT    => FIFO_DATA_PAYLOAD
);

FIFO_DATA_OUT2 <= FIFO_DATA_PAYLOAD(3) &
FIFO_DATA_PAYLOAD(2) & FIFO_DATA_PAYLOAD(1) &
FIFO_DATA_PAYLOAD(0);
FIFO_DATA_OUT3 <= FIFO_DATA_PAYLOAD(7) &
FIFO_DATA_PAYLOAD(6) & FIFO_DATA_PAYLOAD(5) &
FIFO_DATA_PAYLOAD(4);
FIFO_DATA_OUT4 <= X"0000" & FIFO_DATA_PAYLOAD(9) &
FIFO_DATA_PAYLOAD(8);

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if BLOCK_PKT_GOOD = '1' and CMD_NORMAL = '0' and
CMD_NULL = '0' then
     FIFO_WRITE <= '1';
    else
     FIFO_WRITE <= '0';
    end if;

    if( BLOCK_WRITE_L = '0' and BLOCK_ENABLE = '1' and
BLOCK_ADDR_IN( 11 downto 8 ) = X"9" ) then
     FIFO_READ <= '1';
    else
     FIFO_READ <= '0';
    end if;
  end if;
end process;

end Behavioral;
```

9.2.11 DDM_Rx_FIFO_DP.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity ddm_rx_fifo_dp is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  CMD_IN: in T_COMMAND;
  ADDR_TO_IN: in T_NODEADDR;
  ADDR_FROM_IN: in T_NODEADDR;
  DATA_IN: in T_DATA_PAYLOAD;
  NETTIME_IN: in T_NETTIME;

  CMD_OUT: out T_COMMAND;
  ADDR_TO_OUT: out T_NODEADDR;
  ADDR_FROM_OUT: out T_NODEADDR;
  DATA_OUT: out T_DATA_PAYLOAD;
  NETTIME_OUT: out T_NETTIME;

  SIZE: out std_logic_vector( 3 downto 0 );

  R: in std_logic;
  W: in std_logic;

  EMPTY: out std_logic;
  FULL: out std_logic
);
end ddm_rx_fifo_dp;
```

```
architecture Behavioral of ddm_rx_fifo_dp is
type T_PKTREC is record
  CMD: T_COMMAND;
  NETTIME: T_NETTIME;
  TOADDR: T_NODEADDR;
  FROMADDR: T_NODEADDR;
  PAYLOAD: T_DATA_PAYLOAD;
end record;

subtype T_PKTBUF is std_logic_vector( 107 downto 0 );
signal DATA_INX, DATA_OUTX: T_PKTBUF;

constant C_FIFOSIZE: integer := 16;

type T_PKT_ARRAY is array ( 0 to C_FIFOSIZE - 1 ) of
T_PKTBUF;
signal DATA: T_PKT_ARRAY;

signal ADDR_CURR: std_logic_vector( 3 downto 0 );
signal ADDR_NEXT: std_logic_vector( 3 downto 0 );
constant C_ADDR_INIT: std_logic_vector( 3 downto 0 ) :=
"0000";

signal FULL_X: std_logic;
signal EMPTY_X: std_logic;

signal Rp, Wp: std_logic;

signal AC_MINUS_AN: std_logic_vector( 3 downto 0 );
signal AN_MINUS_AC: std_logic_vector( 3 downto 0 );
signal AN_GE_AC: std_logic;
signal SYNC_RD_ADDR: std_logic_vector( 3 downto 0 );

begin
FULL <= FULL_X;
EMPTY <= EMPTY_X;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    AC_MINUS_AN <= ADDR_CURR - ADDR_NEXT;
    AN_MINUS_AC <= ADDR_NEXT - ADDR_CURR;
```

```
    if( ADDR_NEXT >= ADDR_CURR ) then
     AN_GE_AC <= '1';
    else
     AN_GE_AC <= '0';
    end if;

    if( AN_GE_AC = '1' ) then
     SIZE <= AN_MINUS_AC;
    else
     SIZE <= C_FIFOSIZE - AC_MINUS_AN;
    end if;
  end if;
end process;

--Manage pointers to make sure that they do not cross
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     FULL_X <= '0';
     EMPTY_X <= '1';
     Rp <= '0';
     Wp <= '0';
    else
     if( ADDR_NEXT = ADDR_CURR - 1 ) then        -- -1
       FULL_X <= '1';
     else
       FULL_X <= '0';
     end if;

     if( ADDR_NEXT = ADDR_CURR ) then
       EMPTY_X <= '1';
     else
       EMPTY_X <= '0';
     end if;
    end if;

    Rp <= R;
    Wp <= W;
  end if;
end process;
```

```
process( BLOCK_HSCLK ) is
begin
  if(rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     --easier to change size if constant is there
     ADDR_CURR <= C_ADDR_INIT;
     ADDR_NEXT <= C_ADDR_INIT;
    else
     if( R = '1' and Rp = '0' and EMPTY_X = '0') then
       ADDR_CURR <= ADDR_CURR + 1;
     end if;

     if( W = '1' and Wp = '0' and FULL_X = '0' ) then
       ADDR_NEXT <= ADDR_NEXT + 1;
     end if;
    end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if(rising_edge( BLOCK_HSCLK ) ) then
    if( W = '1' and Wp = '0' and FULL_X = '0' ) then
     DATA( CONV_INTEGER(ADDR_NEXT) ) <= DATA_INX;
    end if;

    SYNC_RD_ADDR <= ADDR_CURR;
  end if;
end process;

--DATA_OUTX <= DATA(CONV_INTEGER(SYNC_RD_ADDR));
DATA_OUTX <= DATA(CONV_INTEGER(ADDR_CURR));


DATA_INX(3 downto 0) <= CMD_IN;
DATA_INX(11 downto 4) <= NETTIME_IN;
DATA_INX(19 downto 12) <= ADDR_TO_IN;
DATA_INX(27 downto 20) <= ADDR_FROM_IN;
DATA_INX(35 downto 28) <= DATA_IN(0);
DATA_INX(43 downto 36) <= DATA_IN(1);
DATA_INX(51 downto 44) <= DATA_IN(2);
```

```
DATA_INX(59 downto 52) <= DATA_IN(3);
DATA_INX(67 downto 60) <= DATA_IN(4);
DATA_INX(75 downto 68) <= DATA_IN(5);
DATA_INX(83 downto 76) <= DATA_IN(6);
DATA_INX(91 downto 84) <= DATA_IN(7);
DATA_INX(99 downto 92) <= DATA_IN(8);
DATA_INX(107 downto 100) <= DATA_IN(9);

CMD_OUT <= DATA_OUTX(3 downto 0 );
NETTIME_OUT <= DATA_OUTX(11 downto 4);
ADDR_TO_OUT <= DATA_OUTX(19 downto 12);
ADDR_FROM_OUT <= DATA_OUTX(27 downto 20);
DATA_OUT(0) <= DATA_OUTX(35 downto 28);
DATA_OUT(1) <= DATA_OUTX(43 downto 36);
DATA_OUT(2) <= DATA_OUTX(51 downto 44);
DATA_OUT(3) <= DATA_OUTX(59 downto 52);
DATA_OUT(4) <= DATA_OUTX(67 downto 60);
DATA_OUT(5) <= DATA_OUTX(75 downto 68);
DATA_OUT(6) <= DATA_OUTX(83 downto 76);
DATA_OUT(7) <= DATA_OUTX(91 downto 84);
DATA_OUT(8) <= DATA_OUTX(99 downto 92);
DATA_OUT(9) <= DATA_OUTX(107 downto 100);
end Behavioral;
```

9.2.12 Event.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EVENT_VECTOR is
generic( NVEC: integer := 31 );
port(
  BLOCK_RESET_L: in std_logic;
  BLOCK_HSCLK: in std_logic;
  S: in std_logic_vector( NVEC - 1 downto 0 ); --set
(postiive edge triggered, synchronous)
  R: in std_logic_vector( NVEC - 1 downto 0 );  --reset
(positive edge triggered, synchronous)
  E: out std_logic_vector( NVEC - 1 downto 0 )  --event
out
);
end EVENT_VECTOR;

architecture Behavioral of EVENT_VECTOR is
type T_FSM_EVENT is( FIRE, RESET );
type T_FSM_EVENT_ARRAY is array( 0 to NVEC - 1 ) of
T_FSM_EVENT;
signal STATE, NEXT_STATE: T_FSM_EVENT_ARRAY;

signal Rp, Sp: std_logic_vector( NVEC - 1 downto 0 );
begin

G1: for i in 0 to NVEC - 1 generate
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     Rp(i) <= '0';
     Sp(i) <= '0';
    end if;
    Rp(i) <= R(i);
    Sp(i) <= S(i);
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     STATE(i) <= RESET;
    else
     STATE(i) <= NEXT_STATE(i);
    end if;
  end if;
end process;

process( STATE(i), R(i), S(i), Rp(i), Sp(i) ) is
begin
  case( STATE(i) ) is
    when RESET =>
     E(i) <= '0';

     if( Sp(i) = '0' and S(i) = '1' ) then
       NEXT_STATE(i) <= FIRE;
     else
       NEXT_STATE(i) <= RESET;
     end if;
    when FIRE =>
     E(i) <= '1';

     --If a fire event comes in at the same time as a
reset event,
     --stay in the fire state
     if( Sp(i) = '0' and S(i) = '1' ) then
       NEXT_STATE(i) <= FIRE;
     --If a reset event comes in by itself, reset the
event
     elsif( Rp(i) = '0' and R(i) = '1' ) then
       NEXT_STATE(i) <= RESET;
     else
       NEXT_STATE(i) <= FIRE;
     end if;
    when others =>
```

```
    NEXT_STATE(i) <= RESET;
    E(i) <= '0';
  end case;
end process;
end generate;
end Behavioral;
```

9.2.13 PipeComapre.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PIPE_COMPARE is
generic(
  VECSIZE: integer := 8; RST_VAL: std_logic := '0');
port(
  RESET_L: in std_logic;
  CLK: in std_logic;
  VALUE: in std_logic_vector( VECSIZE - 1 downto 0 );
  TARGET: in std_logic_vector( VECSIZE - 1 downto 0 );
  RESULT: out std_logic);
end PIPE_COMPARE;

architecture Behavioral of PIPE_COMPARE is
--constant TARGET: std_logic_vector( 5 downto 0 ) :=
"000110";
begin

process( CLK ) is
begin
  if( rising_edge(CLK ) ) then
    if( RESET_L = '0' ) then
     RESULT <= '0';
    else
     if( VALUE = TARGET ) then
        RESULT <= '1';
     else
        RESULT <= '0';
     end if;
    end if;
  end if;
end process;

end Behavioral;
```

9.2.14 Deframer.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity deframer is
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;

    RXDATA: in std_logic_vector( 7 downto 0 );
    RXCMD: in std_logic_vector( 3 downto 0 );
    RXSCD: in std_logic;

    VLTN: in std_logic;
    LFI_L: in std_logic;

    RXCLK: in std_logic;
    BLOCK_PKT_GOOD: out std_logic;

    STARTFRAME: out std_logic;

         --From bottom layer
    REDIR_INX_CMD: out T_COMMAND;
    REDIR_INX_NETTIME: out T_NETTIME;
    REDIR_INX_ADDR_TO:  out T_NODEADDR;
    REDIR_INX_ADDR_FROM: out T_NODEADDR;
    REDIR_INX_DATA: out T_DATA_PAYLOAD;
    POSCTR_OUT: out std_logic_vector( 4 downto 0 );


    DEBUG_DEF: out std_logic_vector( 7 downto 0 )
    );
end deframer;

architecture Behavioral of deframer is
component CRCGEN is
  port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  CRC_IN: in std_logic_vector( 7 downto 0 );
  BLOCK_PKT_GOOD: out std_logic;
  CRC_OUT: out std_logic_vector( 7 downto 0 );

  data_in: in T_PACKETBUFFER
  );
end component;


type T_RXBUFFER is array ( 0 to 23 ) of T_BYTE;

signal RXBUFFER: T_RXBUFFER;
signal POSCTR: integer range 0 to 23;
signal POSCTR_INCR_EN: std_logic;
signal TRIP: std_logic;
signal STARTFRAME_TIMEOUT: std_logic;
signal STARTFRAMEX: std_logic;

signal CRC: std_logic_vector( 7 downto 0 );
signal BLOCK_DATA_IN_CRC: T_PACKETBUFFER;
signal BLOCK_PKT_GOODX: std_logic;
signal REDIR_INX_CMDX: std_logic_vector( 3 downto 0 );

signal LFI_TRAP_L: std_logic;
begin

CCRCGEN: CRCGEN port map(
  BLOCK_HSCLK    => CLK,
  BLOCK_RESET_L  => RESET_L,
  CRC_IN    => RXBUFFER(15),
  DATA_IN  => BLOCK_DATA_IN_CRC,
  BLOCK_PKT_GOOD => BLOCK_PKT_GOODX,
```

```
  CRC_OUT   => CRC
);

BLOCK_DATA_IN_CRC(0)  <= "0000" &  REDIR_INX_CMDX;
BLOCK_DATA_IN_CRC(1)  <= RXBUFFER(1);
BLOCK_DATA_IN_CRC(2)  <= RXBUFFER(2);
BLOCK_DATA_IN_CRC(3)  <= RXBUFFER(3);
BLOCK_DATA_IN_CRC(4)  <= RXBUFFER(4);
BLOCK_DATA_IN_CRC(5)  <= RXBUFFER(5);
BLOCK_DATA_IN_CRC(6)  <= RXBUFFER(6);
BLOCK_DATA_IN_CRC(7)  <= RXBUFFER(7);
BLOCK_DATA_IN_CRC(8)  <= RXBUFFER(8);
BLOCK_DATA_IN_CRC(9)  <= RXBUFFER(9);
BLOCK_DATA_IN_CRC(10) <=   RXBUFFER(10);
BLOCK_DATA_IN_CRC(11) <=   RXBUFFER(11);
BLOCK_DATA_IN_CRC(12) <=   RXBUFFER(12);
BLOCK_DATA_IN_CRC(13) <=   RXBUFFER(13);
BLOCK_DATA_IN_CRC(14) <=   RXBUFFER(14);
BLOCK_DATA_IN_CRC(15) <=   X"00";
BLOCK_DATA_IN_CRC(16) <=   X"00";
BLOCK_DATA_IN_CRC(17) <= X"00";
BLOCK_DATA_IN_CRC(18) <= X"00";
BLOCK_DATA_IN_CRC(19) <=   X"00";


STARTFRAME <= STARTFRAMEX;

POSCTR_OUT <= CONV_STD_LOGIC_VECTOR( POSCTR, 5);


--When LFI falls (or dips down), set this signal until
the start of the next packet
process( RESET_L, CLK ) is
begin
  if( rising_edge( CLK ) ) then
    if( RESET_L = '0' ) then
     LFI_TRAP_L <= '1';
    else
     if( RXSCD = '1' ) then
       LFI_TRAP_L <= '1';
     else
       --TRAP LFI as soon as it falls
       LFI_TRAP_L <= LFI_TRAP_L and LFI_L;
     end if;
    end if;
  end if;
end process;

--Detect rising edge on RXCLK
process( RESET_L, CLK ) is
variable LOCK: std_logic;
begin
  if( rising_edge( CLK ) ) then
    if( RESET_L = '0' ) then
     TRIP <= '0';
     LOCK := '0';
    else
     if( RXCLK = '0') then
       TRIP <= '0';
       LOCK := '0';
     elsif( LOCK /= '1' and TRIP = '0' ) then
       TRIP <= '1';
       LOCK := '1';
     else
       TRIP <= '0';
     end if;
    end if;
  end if;
end process;

process( RESET_L, RXCLK ) is
begin
  --Process to select what to transmit
  if( falling_edge( RXCLK ) ) then
    if( RESET_L = '0' ) then
     POSCTR <= 0;
     STARTFRAME_TIMEOUT <= '0';
     STARTFRAMEX <= '0';
     for i in 0 to 22 loop
       RXBUFFER(i) <= X"00";
     end loop;

     BLOCK_PKT_GOOD <= '0';
     REDIR_INX_CMD <= C_CMD_NULL;
```

```
    REDIR_INX_CMDX <= C_CMD_NULL;
   else
      if( POSCTR = 16) then
        STARTFRAMEX <= '1';  --start transmitting a new
frame
      else
        STARTFRAMEX <= '0';
      end if;

      if( POSCTR >= 15) then
        BLOCK_PKT_GOOD <= BLOCK_PKT_GOODX and
LFI_TRAP_L;  --indicate good packet by no LFI and CRC
GOOD
      else
        BLOCK_PKT_GOOD <= '0';
      end if;

      --test for beginning of frame by testing for a
non JK command character
      if( (RXSCD = '1' and RXCMD /= "0000") ) then
        POSCTR <= 1;
        REDIR_INX_CMD <= RXCMD;  --store the command
        REDIR_INX_CMDX <= RXCMD;
        BLOCK_PKT_GOOD <= '0';
        STARTFRAME_TIMEOUT <= '0';
      elsif( POSCTR_INCR_EN = '1' ) then
        POSCTR <= POSCTR + 1;
           STARTFRAME_TIMEOUT <= '0';
      else
        STARTFRAME_TIMEOUT <= '1';
      end if;

      --fill buffer depending on what is RXDATA
      RXBUFFER(POSCTR) <= RXDATA;
   end if;
  end if;
end process;


--Determines whether or not to count
--When counter saturates, this will disable it from
counting
process( RESET_L, CLK ) is
begin
  if( rising_edge( CLK ) ) then
    if( RESET_L = '0' ) then
     POSCTR_INCR_EN <= '0';
    else
     if( POSCTR < 22 ) then
       POSCTR_INCR_EN <= '1';
     else
       POSCTR_INCR_EN <= '0';
     end if;
    end if;
  end if;
end process;

--take data buffer and extract components
process( RESET_L, CLK ) is
begin
  --if reset is active, send sync characters
  if( rising_edge(CLK ) ) then
    if( RESET_L = '0' ) then
     REDIR_INX_ADDR_TO <= X"00";
     REDIR_INX_ADDR_FROM <= X"00";
     REDIR_INX_NETTIME <= X"00";
     REDIR_INX_DATA <= ( X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00");
    else
     REDIR_INX_ADDR_TO <= RXBUFFER(1);
     REDIR_INX_ADDR_FROM <= RXBUFFER(2);
     REDIR_INX_NETTIME <= RXBUFFER(3);
     REDIR_INX_DATA(0) <= RXBUFFER(5);
     REDIR_INX_DATA(1) <= RXBUFFER(6);
     REDIR_INX_DATA(2) <= RXBUFFER(7);
     REDIR_INX_DATA(3) <= RXBUFFER(8);
     REDIR_INX_DATA(4) <= RXBUFFER(9);
     REDIR_INX_DATA(5) <= RXBUFFER(10);
     REDIR_INX_DATA(6) <= RXBUFFER(11);
     REDIR_INX_DATA(7) <= RXBUFFER(12);
     REDIR_INX_DATA(8) <= RXBUFFER(13);
     REDIR_INX_DATA(9) <= RXBUFFER(14);
    end if;
  end if;
```

```
end process;

DEBUG_DEF(7) <= STARTFRAME_TIMEOUT;
DEBUG_DEF(6) <= POSCTR_INCR_EN;
DEBUG_DEF(5) <= STARTFRAMEX;
DEBUG_DEF(4 downto 0 ) <= CONV_STD_LOGIC_VECTOR( POSCTR,
5 );

end Behavioral;
```

9.2.15 CRCGen.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity CRCGEN is
  port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  CRC_IN: in std_logic_vector( 7 downto 0 );
  BLOCK_PKT_GOOD: out std_logic;
  CRC_OUT: out std_logic_vector( 7 downto 0 );

  data_in: in T_PACKETBUFFER
  );
end CRCGEN;

architecture Behavioral of CRCGEN is
signal XOR_L1_W1: std_logic_vector( 7 downto 0);
signal XOR_L1_W2: std_logic_vector( 7 downto 0);
signal XOR_L1_W3: std_logic_vector( 7 downto 0);
--signal XOR_L1_W4: std_logic_vector( 7 downto 0);

signal XOR_L2_W1: std_logic_vector( 7 downto 0 );
constant KEY: std_logic_vector( 7 downto 0 ) := X"73";

begin

process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then

    XOR_L1_W1 <= DATA_IN(0) xor DATA_IN(1) xor DATA_IN(2)
xor DATA_IN(3) xor DATA_IN(4) xor DATA_IN(5) xor
DATA_IN(6) xor DATA_IN(7);
    XOR_L1_W2 <= DATA_IN(8) xor DATA_IN(9) xor
DATA_IN(10) xor DATA_IN(11) xor DATA_IN(12) xor
DATA_IN(13) xor DATA_IN(14) xor DATA_IN(15);
    XOR_L1_W3 <= DATA_IN(16) xor DATA_IN(17) xor
DATA_IN(18) xor DATA_IN(19) xor KEY;

    XOR_L2_W1 <= XOR_L1_W1 + XOR_L1_W2 + XOR_L1_W3;


    CRC_OUT <= XOR_L2_W1;

    if( BLOCK_RESET_L = '0' ) then
     BLOCK_PKT_GOOD <= '0';
    else
     if( XOR_L2_W1 = CRC_IN ) then
       BLOCK_PKT_GOOD <= '1';
     else
       BLOCK_PKT_GOOD <= '0';
     end if;
    end if;
  end if;
end process;

end Behavioral;
```

9.2.16 Event_Man.VHD

```
-- event manager
-- This block communicates between the fault protection
and the command processor

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity event_man is
    Port (
       BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;    -- goes high
when packet is complete
    BLOCK_PKT_GOOD: in std_logic;               --signals
packet is for this block

    BLOCK_PKT_RDY: out std_logic;               -- goes
high when an output packet is ready
    BLOCK_PKT_CLR: in std_logic;         -- clears ready
flag
    FREEZE: in std_logic;                       -- stops
output from changing, while command processor is
clearing ready flag

    BLOCK_CMD_IN: in T_COMMAND;   -- input command
(should always = BLOCK_CMD_IN if packet is for this
block)
    BLOCK_CMD_OUT: out T_COMMAND; -- always =
BLOCK_CMD_IN if for this block

    NET_TIME: in T_NETTIME;             -- current time

    BLOCK_DATA_IN: in T_DATA_PAYLOAD;    -- input
package, BLOCK_DATA_IN(1) = x'01' for fault injection
                                    -- = x'00' for fault
reset and BLOCK_DATA_IN(2) is where the reset vector is
located
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;  --
BLOCK_DATA_OUT(2) is the current fault source vector,
                                                --
BLOCK_DATA_OUT(3) is the current time

    -- fault injection for fault protection testing
    FAULT_INJ: out std_logic;                   --
injects a fault into the fault block
    FAULT_SOURCE_L: in std_logic_vector (4 downto 0); --
current fault sources (from fault block)
    FAULT_RESET_L: out std_logic_vector (4 downto 0);  --
faults to be reset

    PKT_MODE: in E_PACKET_MODE           -- long packet
or short packet
    );
end event_man;

architecture Behavioral of event_man is
  signal FAULT_SOURCE_L_LAST: std_logic_vector (4 downto
0); -- most recent fault source
begin

  -- send fault resets or a fault injection to fault
protection block
  process (BLOCK_HSCLK, BLOCK_RESET_L, BLOCK_CMD_IN,
BLOCK_PKT_COMPLETE)
  begin
    if (rising_edge(BLOCK_HSCLK)) then
    if (BLOCK_RESET_L = '0') then        -- on reset, no
fault injection or fault reset
     FAULT_INJ <= '0';
     FAULT_RESET_L <= "11111";
    elsif (BLOCK_CMD_IN /= C_CMD_EVENT or
BLOCK_PKT_COMPLETE /= '1') then
     FAULT_INJ <= '0';                       -- no fault
injection or fault reset since packet is not ready
```

```
    FAULT_RESET_L <= "11111";

   else
    if (BLOCK_PKT_GOOD = '1') then             -- if
packet is for this block
      if (BLOCK_DATA_IN(1) = x"01") then -- inject a
fault
        FAULT_INJ <= '1';
      else -- process new fault reset
        FAULT_RESET_L <= BLOCK_DATA_IN(2)(4 downto 0);
      end if;
    end if;
   end if;
   end if;
  end process;

  -- detects any change in the fault source and sends
the time to the CMD block
  process (FAULT_SOURCE_L, BLOCK_HSCLK, BLOCK_RESET_L,
PKT_MODE, FREEZE)
  begin
    if (rising_edge(BLOCK_HSCLK)) then
    if( BLOCK_RESET_L = '0' ) then
     FAULT_SOURCE_L_LAST <= "11111"; -- no faults on
reset
     BLOCK_PKT_RDY <= '0';
     BLOCK_CMD_OUT <= C_CMD_EVENT;
     if (PKT_MODE = LONG_PKT) then        -- initialize
outputs
       for i in 0 to 9 loop
         BLOCK_DATA_OUT(i) <= x"00";
       end loop;
     elsif (PKT_MODE = SHORT_PKT) then
       for i in 0 to 3 loop
         BLOCK_DATA_OUT(i) <= x"00";
       end loop;
     end if;

   else
     if (BLOCK_PKT_CLR = '1') then                    --
CMD has received latest packet
       BLOCK_PKT_RDY <= '0';
     -- updates output with latest fault source (if
freeze is not high)
     elsif (FAULT_SOURCE_L_LAST /= FAULT_SOURCE_L and
FREEZE = '0') then
       BLOCK_DATA_OUT(3) <= NET_TIME;
       BLOCK_DATA_OUT(2)(4 downto 0) <= FAULT_SOURCE_L;
       FAULT_SOURCE_L_LAST <= FAULT_SOURCE_L;
       BLOCK_PKT_RDY <= '1';
     end if;
    end if;
    end if;
  end process;
end Behavioral;
```

9.2.17 Extended_Mgr.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;


--Extended manager takes care of management type
protocol functions
--First parameter of payload is instruction word
entity EXTENDED_MGR is
port (
  BLOCK_HSCLK:          in std_logic;
  BLOCK_RESET_L:        in std_logic;

  NET_TIME_OVERRIDE:        out T_NETTIME;
  NET_TIME_SET:         out std_logic;

  BLOCK_PKT_GOOD:       in std_logic;
  BLOCK_PKT_RDY:        out std_logic;
  BLOCK_PKT_CLR:        in std_logic;

  BLOCK_PKT_ADDR_TO_OUT:   out T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_OUT: out T_NODEADDR;
  BLOCK_PKT_CMD_OUT:        out T_COMMAND;
  BLOCK_PKT_DATA_OUT:   out T_DATA_PAYLOAD;

  BLOCK_PKT_ADDR_TO_IN:    in T_NODEADDR;
  BLOCK_PKT_ADDR_FROM_IN:  in T_NODEADDR;
  BLOCK_PKT_CMD_IN:     in T_COMMAND;
  BLOCK_PKT_DATA_IN:        in T_DATA_PAYLOAD;
  SYS_RST_COMM:         out std_logic;
  PIN_RST:       in std_logic;

  LOAD_CLK_VAL:         out std_logic;
  NEW_CLK_VAL:          out T_NETTIME;

  BLOCK_PKT_ABSORB:     out std_logic;

  THIS_ADDR:         out T_NODEADDR       );
end EXTENDED_MGR;

architecture Behavioral of EXTENDED_MGR is
type EXT_MGR_STATE is (idle, flush, setclk, countnodes,
setaddr, sysrst );
signal TX_MGR_STATE, TX_MGR_STATE_NEXT: EXT_MGR_STATE;

component RAND_GEN is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  RAND_OUT: out std_logic_vector( 15 downto 0 )
);
end component;

component PIPE_COMPARE is
generic(
  VECSIZE: integer; RST_VAL: std_logic := '0');
port(
  RESET_L: in std_logic;
  CLK: in std_logic;
  VALUE: in std_logic_vector( VECSIZE - 1 downto 0 );
  TARGET: in std_logic_vector( VECSIZE - 1 downto 0 );
  RESULT: out std_logic);

end component;

--Packet is sent to all nodes in ring.  For each node,
the network time is incremented.
constant C_EX_CMD_SETCLK: T_EX_CMD := X"11";

--Packet is sent to all nodes in ring.  For each node,
the count is incremented.
--Does not modify node's behavior
constant C_EX_CMD_COUNTNODES: T_EX_CMD := X"20";
```

```
signal NODE_COUNT: std_logic_vector( 7 downto 0 );

--Packet is sent to all nodes in ring.  The algorithm
finds the first data field that is zero.
--If that is 2, then it compares it's address to the
address in 1.  If it is the same as this node's address,
--Then it places its address in the 2 field.
--If it is greater than 2, then it appends its address
into that first zero field
--If it is 1, then it places its address there and also
in the second field.
--Packet is forwarded around the ring until it reaches
its master again.
--Does not modify node's behavior
constant C_EX_GETADDR: T_EX_CMD := X"21";

--Sent to a node to set its address.  Packet is
absorbed.
constant C_EX_CMD_SETADDR: T_EX_CMD := X"22";

constant C_EX_CMD_SYS_RST: T_EX_CMD := X"99";
constant C_ADDR_DEFAULT: T_NODEADDR := X"7F";

--Clear the ring of contents.  Do this by sending out
another flushring packet.
--Keep sending null packets for the specified amount of
time
--Format:  [cmd, time...]
--Future: allow option to decrement flush counter for
each node sent to to allow fixed flush time
constant C_EX_CMD_FLUSH_RING: T_EX_CMD := X"30";
signal FLUSH_TIMER: std_logic_vector( 7 downto 0 );
signal FLUSH_MODE: std_logic;  --a 1 indicates in flush
mode, and send only packets of null type
signal FLUSH_TIMER_SAT: std_logic;
signal FLUSH_TIMER_NEXT: std_logic_vector( 7 downto 0 );
signal FLUSH_TIMER_LOAD: std_logic;
signal FLUSH_ABSORB: std_logic;

signal RAND1: std_logic;
signal RAND2: std_logic;
signal RAND_CAPTURE: std_logic_vector( 15 downto 0 );

signal SEM_REQUEST: std_logic;  --asking for semaphore
signal SEM_HOLD: std_logic;        --have semaphore



constant C_ADDR_BROADCAST: T_NODEADDR := X"00";

type T_FSM_SYS_RST is ( idle, pending, active );
signal FSM_SYS_RST: T_FSM_SYS_RST;
--constant SYS_RST_DELAY_MAX: std_logic_vector( 31
downto 0 ) := X"04C4B400";  --1 second
--constant SYS_RST_HOLD_MAX: std_logic_vector( 31 downto
0 )  := X"04C4B400";  --1 second

--For simulation purposes, speed up these counters
constant SYS_RST_DELAY_MAX: std_logic_vector( 15 downto
0 ) := X"04C4";  --1 second
constant SYS_RST_HOLD_MAX: std_logic_vector( 15 downto 0
)  := X"04C4";  --1 second


signal SYS_RST_DELAY_CTR: std_logic_vector( 15 downto 0
);
signal SYS_RST_HOLD_CTR: std_logic_vector( 15 downto 0
);
signal SYS_RST: std_logic;
signal CMD_EX_SYS_RST: std_logic;
signal SYS_RST_FORWARD: std_logic;
signal SYS_RST_NO_RST_SELF: std_logic;



--Addressing signals
signal ADDR_THIS: std_logic;
signal ADDR_BROADCAST: std_logic;
signal ADDR_THIS_OR_BROADCAST: std_logic;
signal FROM_THIS: std_logic;

signal CMD_EXTENDED: std_logic;

signal CMD_SETCLK: std_logic;
```

```
signal CMD_SETADDR: std_logic;
signal CMD_FLUSHRING: std_logic;

signal SETCLK_TIME: T_NETTIME;
signal SETCLK_FORWARD: std_logic;

signal FLUSH_TIMER_SAT_AND_BLOCK_PKT_CLR: std_logic;
signal THIS_ADDRX: T_NODEADDR;
begin

--only change block pkt good on falling edge
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK )  ) then
    if( BLOCK_PKT_GOOD = '0' ) then
     THIS_ADDR <= THIS_ADDRX;
    end if;
  end if;
end process;


SYS_RST_NO_RST_SELF <= BLOCK_PKT_DATA_IN(1)(0);
--This process initiates a system reset of the current
FPGA
process( BLOCK_HSCLK ) is

begin
  if( rising_edge( BLOCK_HSCLK )) then
    if( CMD_EX_SYS_RST = '1' and BLOCK_PKT_GOOD = '1' and
CMD_EXTENDED = '1' and ADDR_THIS_OR_BROADCAST = '1' )
then
     if( ADDR_THIS = '1' ) then
       SYS_RST_FORWARD <= '0';    --forward packet if it
isn't to this node only
       SYS_RST <= not SYS_RST_NO_RST_SELF;  --if not set
to block self-reset, do it
     else
       SYS_RST_FORWARD <= '1';    --forward packet if it
isn't to this node only
       SYS_RST <= '1';
     end if;
    else
     SYS_RST <= '0';
     SYS_RST_FORWARD <= '0';
    end if;

    --if the pin is pushed for reset (from FPGA input)
    if( PIN_RST = '0' ) then
     FSM_SYS_RST <= idle;
     SYS_RST_HOLD_CTR <= X"0000";
     SYS_RST_DELAY_CTR <= X"0000";
    else
     case( FSM_SYS_RST ) is
     when idle =>
       SYS_RST_DELAY_CTR <= X"0000";
       SYS_RST_HOLD_CTR <= X"0000";
       SYS_RST_COMM <= '0';

       if( SYS_RST = '1' ) then
         FSM_SYS_RST <= pending;
       else
         FSM_SYS_RST <= idle;
       end if;
     when pending =>
       SYS_RST_COMM <= '0';
       SYS_RST_DELAY_CTR <= SYS_RST_DELAY_CTR + 1;
       SYS_RST_HOLD_CTR <= X"0000";

       if( SYS_RST_DELAY_CTR >= SYS_RST_DELAY_MAX ) then
         FSM_SYS_RST <= active;
       else
         FSM_SYS_RST <= pending;
       end if;
     when active =>
       SYS_RST_COMM <= '1';
       SYS_RST_DELAY_CTR <= X"0000";
       SYS_RST_HOLD_CTR <= SYS_RST_HOLD_CTR + 1;

       if( SYS_RST_HOLD_CTR >= SYS_RST_HOLD_MAX ) then
         FSM_SYS_RST <= idle;
       else
         FSM_SYS_RST <= active;
       end if;
     when others =>
```

```
      FSM_SYS_RST <= idle;
      SYS_RST_DELAY_CTR <= X"0000";
      SYS_RST_COMM <= '0';
      SYS_RST_HOLD_CTR <= X"0000";
    end case;
   end if;

  end if;
end process;


process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    RAND_CAPTURE(15 downto 1 ) <= RAND_CAPTURE(14 downto
0);
    RAND_CAPTURE(0) <= RAND_CAPTURE(15) xor RAND1;
  end if;
end process;

--RAND1 <= not (not (not RAND2) );
--RAND2 <= RAND1;

--  ADDR_THIS = ( BLOCK_PKT_ADDR_TO_IN = THIS_ADDRX )
PIPE_COMPARE1: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_ADDR_TO_IN,
      TARGET => THIS_ADDRX, RESULT => ADDR_THIS );

--  ADDR_BROADCAST = ( BLOCK_PKT_ADDR_TO_IN =
ADDR_BROADCAST )
PIPE_COMPARE2: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_ADDR_TO_IN,
      TARGET => C_ADDR_BROADCAST, RESULT =>
ADDR_BROADCAST );

--  FROM_THIS = ( BLOCK_PKT_ADDR_FROM_IN = THIS_ADDRX )
PIPE_COMPARE3: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_ADDR_FROM_IN,
      TARGET => THIS_ADDRX, RESULT => FROM_THIS );

--  CMD_SETCLK = ( BLOCK_PKT_DATA_IN(0) =
C_EX_CMD_SETCLK )
PIPE_COMPARE4: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_DATA_IN(0),
      TARGET => C_EX_CMD_SETCLK, RESULT => CMD_SETCLK
);

--  CMD_FLUSHRING = ( BLOCK_PKT_DATA_IN(0) =
C_EX_CMD_FLUSH_RING )
PIPE_COMPARE5: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_DATA_IN(0),
      TARGET => C_EX_CMD_FLUSH_RING, RESULT =>
CMD_FLUSHRING );

--  CMD_FLUSHRING = ( BLOCK_PKT_DATA_IN(0) =
C_EX_CMD_FLUSH_RING )
PIPE_COMPARE6: PIPE_COMPARE generic map (VECSIZE => 4)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_CMD_IN,
      TARGET => C_CMD_EXTENDED, RESULT => CMD_EXTENDED
);

PIPE_COMPARE8: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_DATA_IN(0),
      TARGET => C_EX_CMD_SETADDR, RESULT => CMD_SETADDR
);

PIPE_COMPARE9: PIPE_COMPARE generic map (VECSIZE => 8)
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_PKT_DATA_IN(0),
      TARGET => C_EX_CMD_SYS_RST, RESULT =>
CMD_EX_SYS_RST );


--Process sets flags for address to 1 if addressed to
this node or broadcasting
process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
```

```
  --set specific command flags
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     ADDR_THIS_OR_BROADCAST <= '0';
    else
     ADDR_THIS_OR_BROADCAST <= ADDR_THIS or
ADDR_BROADCAST;
    end if;
  end if;
end process;

--#########################################################
-- Setclk
-- Netclk receptor process --Format of extended data is
[ECMD,  NETTIME, xx,xx,....]
--#########################################################
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     LOAD_CLK_VAL <= '0';
     NEW_CLK_VAL <= X"00";
    else
     if( BLOCK_PKT_GOOD = '1' and CMD_SETCLK = '1' and
CMD_EXTENDED = '1' ) then
       NEW_CLK_VAL <= BLOCK_PKT_DATA_IN(1);
       LOAD_CLK_VAL  <= '1';
     else
       LOAD_CLK_VAL <= '0';
     end if;
    end if;
  end if;
end process;

process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    SETCLK_TIME <= BLOCK_PKT_DATA_IN(1)+1;

    --if it isn't from this node, forward the packet on
    if( FROM_THIS = '0' and CMD_EXTENDED = '1' ) then
     SETCLK_FORWARD <= '1';
    else
     SETCLK_FORWARD <= '0';
    end if;
  end if;
end process;


--#########################################################
--SetAddr
--If this packet is received, set your node to this
address (quick way for now)
--#########################################################
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     THIS_ADDRX <= C_ADDR_DEFAULT;
    else
     if( BLOCK_PKT_GOOD = '1' and ADDR_THIS = '1' and
CMD_SETADDR = '1' and CMD_EXTENDED = '1') then
       THIS_ADDRX <= BLOCK_PKT_DATA_IN(1);
     end if;
    end if;
  end if;
end process;

--
#########################################################
--Flush Ring
--If this packet is received, set your node to this
address (quick way for now)
--#########################################################
--  FLUSH_SAT = ( FLUSH_TIMER = X"00" )  --counting down
PIPE_COMPARE7: PIPE_COMPARE generic map (VECSIZE => 8,
RST_VAL => '1')
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => FLUSH_TIMER,
       TARGET => X"00", RESULT => FLUSH_TIMER_SAT );

--#########################################################
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable FL_LOCKOUT: std_logic;
```

```
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L=  '0' ) then
     FLUSH_TIMER <= X"00";
     FL_LOCKOUT := '0';
  --   FLUSH_TIMER_SAT <= '1';
     FLUSH_MODE <= '0';
     FLUSH_ABSORB <= '0';
    else
     --If it is a good packet,
     --And this node is supposed to do something with it
     --And this node didn't send it
     --and it is telling fpga to flush ring
     if( BLOCK_PKT_GOOD = '1' and ADDR_THIS_OR_BROADCAST
= '1' and FROM_THIS = '0' and CMD_FLUSHRING = '1' and
CMD_EXTENDED = '1') then
       FLUSH_TIMER <= BLOCK_PKT_DATA_IN(1);
       FLUSH_MODE <= '1';
       FLUSH_TIMER_LOAD <= '1';
     elsif( FLUSH_TIMER_SAT = '1' ) then
       FLUSH_MODE <= '0';
     else
       FLUSH_TIMER_LOAD <= '0';
     end if;

     --If we are gettig our own packet back (FROM_THIS =
'1') and it is a flush commad, we should absorb it back
     FLUSH_ABSORB <= CMD_FLUSHRING and BLOCK_PKT_GOOD and
FROM_THIS;

     if( FLUSH_TIMER_LOAD <= '1' ) then
       FLUSH_TIMER_NEXT <= BLOCK_PKT_DATA_IN(1);
     else
       FLUSH_TIMER_NEXT <= FLUSH_TIMER - 1;
     end if;

     FLUSH_TIMER_SAT_AND_BLOCK_PKT_CLR <= (not
FLUSH_TIMER_SAT) and BLOCK_PKT_CLR;
     --count the number of times a packet is sent
(BLOCK_PKT_CLR transitions to 1)
     if( FLUSH_MODE = '1' and FL_LOCKOUT = '0' and
FLUSH_TIMER_SAT_AND_BLOCK_PKT_CLR = '1' ) then
       FLUSH_TIMER <= FLUSH_TIMER_NEXT;
       FL_LOCKOUT := '1';
     elsif( FLUSH_TIMER_LOAD = '1' ) then
       FLUSH_TIMER <= FLUSH_TIMER_NEXT;
       FL_LOCKOUT := '0';
     elsif( BLOCK_PKT_CLR = '0' ) then
       FL_LOCKOUT := '0';
     end if;
    end if;
  end if;
end process;

--#########################################################
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     TX_MGR_STATE <= idle;
    else
     case( TX_MGR_STATE ) is
     when idle =>
       if( BLOCK_PKT_GOOD = '1' and CMD_EXTENDED = '1' )
then
         if( FLUSH_MODE = '1' ) then
           TX_MGR_STATE <= flush;
         elsif( CMD_SETCLK = '1' ) then
           TX_MGR_STATE <= setclk;
         elsif( CMD_EX_SYS_RST = '1' ) then
           TX_MGR_STATE <= sysrst;
         end if;
       else
         TX_MGR_STATE <= idle;
       end if;
     when flush =>
       if( FLUSH_MODE = '0') then
         TX_MGR_STATE <= idle;
       else
         TX_MGR_STATE <= flush;
       end if;
     when setclk =>
       if( BLOCK_PKT_CLR = '1' ) then
         TX_MGR_STATE <= idle;
```

```
      else
        TX_MGR_STATE <= setclk;
      end if;
    when sysrst =>
      if( BLOCK_PKT_CLR = '1' ) then
        TX_MGR_STATE <= idle;
      else
        TX_MGR_STATE <= sysrst;
      end if;
    when others =>
      TX_MGR_STATE <= idle;
    end case;
   end if;
  end if;
end process;


--#########################################################
--countnodes
--#########################################################
process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     NODE_COUNT <= X"00";
    else
     NODE_COUNT <= BLOCK_PKT_DATA_IN(1) + 1;
    end if;
  end if;
end process;

--#########################################################
process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     BLOCK_PKT_ABSORB <= '0';
    else
     if( FLUSH_MODE = '1' ) then
       BLOCK_PKT_ABSORB <= '1';
     elsif( CMD_SETADDR = '1' and CMD_EXTENDED = '1' )
then
       BLOCK_PKT_ABSORB <= '1';
     else
       BLOCK_PKT_ABSORB <= '0';
     end if;
    end if;
  end if;
end process;


--#########################################################
--state to manage outputs to command manager
--#########################################################
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    case( TX_MGR_STATE ) is
     when idle =>
       BLOCK_PKT_RDY <= '0';
       BLOCK_PKT_ADDR_TO_OUT <= X"00";
       BLOCK_PKT_ADDR_FROM_OUT <= THIS_ADDRX;
       BLOCK_PKT_CMD_OUT <= C_CMD_NULL;
       BLOCK_PKT_DATA_OUT <= (X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00");
     when flush =>
       BLOCK_PKT_RDY <= not BLOCK_PKT_CLR;  --cause to
handshake with cmd processor
       BLOCK_PKT_ADDR_TO_OUT <= X"00";
       BLOCK_PKT_ADDR_FROM_OUT <= THIS_ADDRX;
       BLOCK_PKT_CMD_OUT <= C_CMD_NULL;
       BLOCK_PKT_DATA_OUT <= (X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00");
     when countnodes =>
       BLOCK_PKT_RDY <= not BLOCK_PKT_CLR;
       BLOCK_PKT_ADDR_TO_OUT <= X"00";
       BLOCK_PKT_ADDR_FROM_OUT <=
BLOCK_PKT_ADDR_FROM_IN;
       BLOCK_PKT_CMD_OUT <= C_CMD_EXTENDED;
       BLOCK_PKT_DATA_OUT <= (C_EX_CMD_COUNTNODES,
NODE_COUNT, X"00", X"00", X"00",
              X"00", X"00", X"00", X"00", X"00");
     when setclk =>
       if( SETCLK_FORWARD = '1' ) then
```

```
        BLOCK_PKT_RDY <= not BLOCK_PKT_CLR;
       else
        BLOCK_PKT_RDY <= '0';
       end if;

       BLOCK_PKT_ADDR_TO_OUT <= BLOCK_PKT_ADDR_TO_IN;
       BLOCK_PKT_ADDR_FROM_OUT <=
BLOCK_PKT_ADDR_FROM_IN;
       BLOCK_PKT_CMD_OUT <= C_CMD_EXTENDED;
       BLOCK_PKT_DATA_OUT <= (X"00", X"00", X"00",
X"00", X"00",
             X"00", X"00", X"00", SETCLK_TIME,
C_EX_CMD_SETCLK);

    when sysrst =>
       if( SYS_RST_FORWARD = '1' ) then
        BLOCK_PKT_RDY <= not BLOCK_PKT_CLR;
       else
        BLOCK_PKT_RDY <= '0';
       end if;

       BLOCK_PKT_ADDR_TO_OUT <= BLOCK_PKT_ADDR_TO_IN;
       BLOCK_PKT_AddR_FROM_OUT <=
BLOCK_PKT_ADDR_FROM_IN;
       BLOCK_PKT_CMD_OUT <= C_CMD_EXTENDED;
       BLOCK_PKT_DATA_OUT <= (RAND_CAPTURE(7 downto 0 ),
X"00", X"00", X"00", X"00",
             X"00", X"00", X"00", "0000000" &
SYS_RST_NO_RST_SELF, C_EX_CMD_SYS_RST);
    when others => null;
   end case;
  end if;
end process;

end Behavioral;
```

9.2.18 Framer.VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity framer is
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;

    --Interface to CY7C9689A-AC transmit side
    TXDATA: out std_logic_vector( 7 downto 0 );
    TXCMD: out std_logic_vector( 3 downto 0 );
    TXSCD: out std_logic;
    REFCLK: in std_logic;
    TXFULL_L: in std_logic;

    --Indication to start sending new frame
    STARTFRAME: in std_logic;
--   REQ_DATA: in std_logic;

    --Packet data to use when sending frame
    REDIR_OUTX_CMD: in T_COMMAND;
    REDIR_OUTX_ADDR_TO:  in T_NODEADDR;
    REDIR_OUTX_ADDR_FROM: in T_NODEADDR;
    REDIR_OUTX_DATA: in T_DATA_PAYLOAD;
    NETTIME: in T_NETTIME;
    POSCTR_OUT: out std_logic_vector( 4 downto 0 );

    COPY_PKT: out std_logic;
    PKT_RDY: in std_logic;

    DEBUG_FRM: out std_logic_vector( 7 downto 0 )

    );
end framer;

architecture Behavioral of framer is

component CRCGEN is
  port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  CRC_IN: in std_logic_vector( 7 downto 0 );
  BLOCK_PKT_GOOD: out std_logic;
  CRC_OUT: out std_logic_vector( 7 downto 0 );

  data_in: in T_PACKETBUFFER
  );
end component;

type T_FSM_COPY is ( waiting, active, done );
signal FSM_STATE_LOADER: T_FSM_COPY;

--Determines which byte of the packet is being sent
signal POSCTR: std_logic_vector( 4 downto 0 );
constant POSCTR_SAT: std_logic_vector( 4 downto 0 )  :=
CONV_STD_LOGIC_VECTOR( 17 ,5 );


--Detects the falling edge of the character clock
signal TRIP: std_logic;

--Detects rising edge of start frame to latch new data
and reset counter
signal STARTFRAMEX: std_logic;  --oneshot version of
startframe

--These signals capture frame that is being sent
--They are latched at the start of the frame
signal REDIR_OUTX_CMDX:  T_COMMAND;
signal REDIR_OUTX_ADDR_TOX:   T_NODEADDR;
signal REDIR_OUTX_ADDR_FROMX:  T_NODEADDR;
```

```
signal REDIR_OUTX_DATAX:  T_DATA_PAYLOAD;

signal REDIR_OUTX_CMDX2:  T_COMMAND;
signal REDIR_OUTX_ADDR_TOX2:   T_NODEADDR;
signal REDIR_OUTX_ADDR_FROMX2:  T_NODEADDR;
signal REDIR_OUTX_DATAX2:  T_DATA_PAYLOAD;


signal CRC:  std_logic_vector( 7 downto 0 );
signal BLOCK_DATA_IN_CRC: T_PACKETBUFFER;

signal NEXTPOS: std_logic_vector( 4 downto 0 );
signal SFTRIGGER: std_logic;

signal NETTIMEX, NETTIMEX2, NETTIME_P1: T_NETTIME;
           --17
constant C_POSCTR_GETNEW: integer :=  17;
signal COPY_PKTX: std_logic;
--signal COPY_CTR: std_logic_vector( 1 downto 0 );
begin

COPY_PKT <= COPY_PKTX;

CCRCGEN: CRCGEN port map(
  BLOCK_HSCLK    => CLK,
  BLOCK_RESET_L  => RESET_L,
  CRC_IN    => "00000000",
  DATA_IN  => BLOCK_DATA_IN_CRC,
  CRC_OUT   => CRC
);

BLOCK_DATA_IN_CRC(0)  <= "0000" &  REDIR_OUTX_CMDX;
BLOCK_DATA_IN_CRC(1)  <= REDIR_OUTX_ADDR_TOX;
BLOCK_DATA_IN_CRC(2)  <= REDIR_OUTX_ADDR_FROMX;
BLOCK_DATA_IN_CRC(3)  <= NETTIMEX;
BLOCK_DATA_IN_CRC(4)  <= X"00";
BLOCK_DATA_IN_CRC(5)  <= REDIR_OUTX_DATAX(0);
BLOCK_DATA_IN_CRC(6)  <= REDIR_OUTX_DATAX(1);
BLOCK_DATA_IN_CRC(7)  <= REDIR_OUTX_DATAX(2);
BLOCK_DATA_IN_CRC(8)  <= REDIR_OUTX_DATAX(3);
BLOCK_DATA_IN_CRC(9)  <= REDIR_OUTX_DATAX(4);
BLOCK_DATA_IN_CRC(10) <=   REDIR_OUTX_DATAX(5);
BLOCK_DATA_IN_CRC(11) <=   REDIR_OUTX_DATAX(6);
BLOCK_DATA_IN_CRC(12) <=   REDIR_OUTX_DATAX(7);
BLOCK_DATA_IN_CRC(13) <=   REDIR_OUTX_DATAX(8);
BLOCK_DATA_IN_CRC(14) <=   REDIR_OUTX_DATAX(9);
BLOCK_DATA_IN_CRC(15) <=   X"00";
BLOCK_DATA_IN_CRC(16) <=   X"00";
BLOCK_DATA_IN_CRC(17) <= X"00";
BLOCK_DATA_IN_CRC(18) <= X"00";
BLOCK_DATA_IN_CRC(19) <=   X"00";

PX2: process( RESET_L, CLK ) is
begin
  if( rising_Edge( CLK ) )then
    if RESET_L = '0' then
     REDIR_OUTX_CMDX2 <= C_CMD_NULL;
     REDIR_OUTX_ADDR_TOX2 <= X"00";
     REDIR_OUTX_ADDR_FROMX2 <= X"00";
     REDIR_OUTX_DATAX2 <= (X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00" );
     NETTIMEX2 <= X"00";
     FSM_STATE_LOADER <= waiting;
    else
       case( FSM_STATE_LOADER ) is
       when waiting =>
         if( PKT_RDY = '1' ) then
           FSM_STATE_LOADER <= active;

           REDIR_OUTX_CMDX2        <= REDIR_OUTX_CMD;
           REDIR_OUTX_ADDR_TOX2    <= REDIR_OUTX_ADDR_TO;
           REDIR_OUTX_ADDR_FROMX2 <=REDIR_OUTX_ADDR_FROM;
           REDIR_OUTX_DATAX2       <= REDIR_OUTX_DATA;
           NETTIMEX2               <= NETTIME_P1;

         else
           FSM_STATE_LOADER <= waiting;
         end if;
       when active =>
         FSM_STATE_LOADER <= done;
         REDIR_OUTX_CMDX2         <= REDIR_OUTX_CMD;
         REDIR_OUTX_ADDR_TOX2     <= REDIR_OUTX_ADDR_TO;
         REDIR_OUTX_ADDR_FROMX2   <=REDIR_OUTX_ADDR_FROM;
         REDIR_OUTX_DATAX2        <= REDIR_OUTX_DATA;
```

```
        NETTIMEX2                <= NETTIME_P1;
       when done =>
        if( COPY_PKTX = '0' and PKT_RDY = '0') then
          FSM_STATE_LOADER <= waiting;
        else
          FSM_STATE_LOADER <= done;
        end if;
       when others =>
        FSM_STATE_LOADER <= waiting;
       end case;
    end if;
  end if;
end process;



PCOPY_PKT: process( RESET_L, CLK ) is
begin
  if( rising_edge( CLK ) ) then
    if( RESET_L = '0' ) then
     COPY_PKTX <= '0';
    else
     if( POSCTR =  C_POSCTR_GETNEW ) then
       COPY_PKTX <= '1';
     elsif( FSM_STATE_LOADER = done ) then
       COPY_PKTX <= '0';
     end if;
    end if;
  end if;
end process;

--Detect falling edge on RXCLK
PTRIP: process( RESET_L, CLK ) is
--If edge is already detected, lock out the reset on the
trip signal
variable LOCK: std_logic;
begin
  if( rising_edge( CLK ) ) then

    if( RESET_L = '0' ) then
     TRIP <= '0';
     LOCK := '0';
    else
     --REFCLK is in the positive state, reset the lock
and wait for transition
     --*** Note - may have to use TXFULL_L (lesson
learned from previous pesnet 1.2 design)
     if( TXFULL_L = '1') then
       TRIP <= '0';
       LOCK := '0';
     --REFCLK has transitioned.  Set trip to indicate
this is the clock after the transition is detected.
     elsif( LOCK /= '1' and TRIP = '0' ) then
       TRIP <= '1';
       LOCK := '1';
     --REFCLK has already transitioned, reset trip and
wait for it  to go high again
     else
       TRIP <= '0';
     end if;
    end if;
  end if;
end process;


--Detect rising edge of startframe
PX: process( RESET_L, CLK ) is
variable LOCKSF: std_logic;
begin
  if( rising_edge( CLK ) ) then
  if(RESET_L = '0') then
    STARTFRAMEX <= '0';
    LOCKSF := '0';
    REDIR_OUTX_CMDX <= C_CMD_NULL;
    REDIR_OUTX_ADDR_TOX <= X"00";
    REDIR_OUTX_ADDR_FROMX <= X"00";
    NETTIMEX <= X"00";
    REDIR_OUTX_DATAX <= (X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00" );
  else
    --Wait for the start of a frame
    if( STARTFRAME = '0') then
     STARTFRAMEX <= '0';
```

```
    LOCKSF := '0';
   --As soon as STARTFRAME rises, sample the inputs and
lock them in
   --then wait for STARTFRAME to go low
   --Pulse STARTFRAMEX to reset position counter
   elsif( LOCKSF /= '1' and STARTFRAME = '1' ) then
    STARTFRAMEX <= '1';
    LOCKSF := '1';

    --lock in new packet
    REDIR_OUTX_CMDX       <= REDIR_OUTX_CMDX2;
    REDIR_OUTX_ADDR_TOX   <=REDIR_OUTX_ADDR_TOX2;
    REDIR_OUTX_ADDR_FROMX <= REDIR_OUTX_ADDR_FROMX2;
    REDIR_OUTX_DATAX      <= REDIR_OUTX_DATAX2;
    NETTIMEX          <= NETTIMEX2;
   else
    STARTFRAMEX <= '0';
   end if;
  end if;
  end if;
end process;

--This process manages the position counter.
--STARTFRAMEX resets it
--Trip will enable it to be changed.  If STARTFRAME is 1
then it will reset to 0.  Otherwise it will count,
saturating at
--a maximum to prevent it from rolling over.
PPOSCTR: process( RESET_L, CLK ) is
--variable NEXTPOS: std_logic_vector( 4 downto 0 );
variable LOCKPOS: std_logic;
--variable SFTRIGGER: std_logic;
begin
  --Process to select what to transmit
  if( rising_edge( CLK ) ) then
     if( RESET_L = '0' ) then
    POSCTR <= "00000";
    NEXTPOS <= "00000";
    SFTRIGGER <= '0';
    LOCKPOS := '0';
  else
    if( STARTFRAME = '1' ) then
     NEXTPOS <= "00000";
     SFTRIGGER <= '1'; --lockout the increment
    elsif( POSCTR = X"00000" ) then  --position was reset
     SFTRIGGER <= '0';
     NEXTPOS <= POSCTR + 1;
    elsif( POSCTR < POSCTR_SAT and SFTRIGGER /= '1') then
     NEXTPOS <= POSCTR + 1;
    end if;

    if( TXFULL_L = '0' and LOCKPOS = '0' ) then
     POSCTR <= NEXTPOS;
     LOCKPOS := '1';
    elsif( TXFULL_L = '1' ) then
     LOCKPOS := '0';
    end if;

  end if;
  end if;
end process;

POSCTR_OUT <= POSCTR;

--Determine which part of packet to send based on the
POSCTR variable
PTX: process( RESET_L, CLK ) is
--variable NETTIME_P1: T_NETTIME;
begin
  --if reset is active, send sync characters
  if( rising_edge(CLK ) ) then
    if( RESET_L = '0' ) then
     TXSCD <= '1';
     TXDATA <= X"00";
     TXCMD <= X"0";
     NETTIME_P1 <= X"00";
    else
     case POSCTR is
     when "00000" =>  --transmit command
       TXSCD <= '1';
       TXCMD <= REDIR_OUTX_CMDX;
       TXDATA <= X"00";
     when "00001" => --transmit to address
       TXSCD <= '0';
```

```
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_ADDR_TOX;
      when "00010" => --transmit from address
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_ADDR_FROMX;
      when "00011" => --transmit time
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= NETTIMEX; -- NETTIME_P1;  --temporary
time
      when "00100" => --transmit fltaddr
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= X"00";  --temporary fault address
      when "00101" => --transmit data0
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(0);
      when "00110" => --transmit data1
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(1);
      when "00111" => --transmit data2
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(2);
      when "01000" => --transmit data3
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(3);
      when "01001" => --transmit data4
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(4);
      when "01010" => --transmit data5
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(5);
      when "01011" => --transmit data6
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(6);
      when "01100" => --transmit data7
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(7);
      when "01101" => --transmit data8
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(8);
      when "01110" => --transmit data9
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= REDIR_OUTX_DATAX(9);
      when "01111" => --transmit data9
        TXSCD <= '0';
        TXCMD <= X"0";
        TXDATA <= CRC;   --temporary crc

      when others =>
        TXSCD <= '1';
        TXCMD <= X"0";
        TXDATA <= X"00";
      end case;

      NETTIME_P1 <= NETTIME + 1;
     end if;
   end if;
end process;

DEBUG_FRM(7 downto 4) <= REDIR_OUTX_CMDX2;

DEBUG_FRM(0) <= '1' when FSM_STATE_LOADER = waiting else
'0';
DEBUG_FRM(1) <= '1' when FSM_STATE_LOADER = active else
'0';
DEBUG_FRM(2) <= '1' when FSM_STATE_LOADER = done else
'0';
DEBUG_FRM(3) <= PKT_RDY;
);


end Behavioral;
```

9.2.19 IndexMgr.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity indexmgr is
  port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  BLOCK_PKT_GOOD: in std_logic;
  START_FRAME: out std_logic;
  IN_POSCTR: in std_logic_vector( 4 downto 0 );
  OUT_POSCTR: in std_logic_vector( 4 downto 0 );
  PKT_LENGTH: in std_logic_vector( 4 downto 0 );
  PKT_ALIGNMENT: in std_logic_vector( 4 downto 0 );

  OPEN_LOOP_MODE: in std_logic;

  TXCLK: in std_logic;
  RXCLK: in std_logic);
end indexmgr;

architecture Behavioral of indexmgr is

component WATCHDOG is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  WDT_EXPIRE: out std_logic;
  R: in std_logic;        --reset (edge sensitive,
synchronous)
  C: in std_logic;        --count (edge sensitive,
synchronous)

  WDT_THRESHOLD: in std_logic_vector( 7 downto 0 )
);
end component;

--signal PER_MAX: integer range 0 to 31;
signal PER_CALC:  std_logic_vector( 4 downto 0 );

signal RX_GOOD_PKT_TIME: integer range 0 to 31;
constant PER_TRIGGER: std_logic_vector( 4 downto 0 ) :=
"01001";
signal WDT_FIRE: std_logic;
signal START_FRAMEX: std_logic;

constant PKT_LENGTH_MIN : std_logic_vector( 4 downto 0 )
:= CONV_STD_LOGIC_VECTOR( 17, 5 );
signal PKT_LENGTH_ACTUAL: std_logic_vector( 4 downto 0
);

begin
--PER_MAX <= 18;  --for now, put this as a constant.  it
may become variable in the future
        -- depending on how to compensate for a slip in
characters

--clamp the minimum packet length
process( BLOCK_HSCLK  ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( CONV_INTEGER(PKT_LENGTH) >=
CONV_INTEGER(PKT_LENGTH_MIN) ) then
     PKT_LENGTH_ACTUAL <= PKT_LENGTH;
    else
     PKT_LENGTH_ACTUAL <= PKT_LENGTH_MIN;
    end if;
  end if;
end process;



--If a frame hasn't been started for a while, fire
watchdog and transmit a packet
C_WATCHDOG: WATCHDOG port map(
  BLOCK_HSCLK => BLOCK_HSCLK,
```

```
  BLOCK_RESET_L => BLOCK_RESET_L,

  WDT_EXPIRE => WDT_FIRE,
  R => START_FRAMEX,
  C => TXCLK,
  WDT_THRESHOLD => "10011100"
);

START_FRAME <= START_FRAMEX;
--set schedule for transmitter
process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable LOCK: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     LOCK := '0';
     PER_CALC <= "00000";
     START_FRAMEX <= '0';
    else
     if( TXCLK = '0' ) then
       LOCK := '0';
       START_FRAMEX <= '0';
     --look for a rising edge on the transmit clock
     elsif( TXCLK = '1' and LOCK = '0' ) then
       LOCK := '1';
       --if a packet is coming in, and it is at the
right point in that received packet,
       --align the transmit timer with this event
       if( IN_POSCTR = PER_TRIGGER and OPEN_LOOP_MODE =
'0') then
         PER_CALC <= "00000";
       --keep counting until it saturates
       elsif( PER_CALC < PKT_LENGTH_ACTUAL ) then
         PER_CALC <= PER_CALC + 1;
       else
         --and then start over, regardless of input
stream
         PER_CALC <= "00000";
       end if;
       if( PER_CALC = PKT_ALIGNMENT or WDT_FIRE = '1' )
then
         START_FRAMEX <= '1';
       else
         START_FRAMEX <= '0';
       end if;
     end if;
    end if;
  end if;
end process;

--what is the best way to time a packet?  (Imagine
packet is deformed and may contain
--more than 1 start frame.  In that case, what is the
definition of a deformed packet?
--which cases of deformed packets matter?  what about
one with a command in the middle?
--I think that a packet size can be measured from the
first command character until the next
--command character that follows.
process( RXCLK, BLOCK_HSCLK, BLOCK_RESET_L ) is
variable GOOD_POSITION_LOCKOUT: std_logic;
begin
  if( rising_edge( RXCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     RX_GOOD_PKT_TIME <= 0;
     GOOD_POSITION_LOCKOUT := '0';
    else
     if( BLOCK_PKT_GOOD = '1' and GOOD_POSITION_LOCKOUT =
'0' ) then
       --we know that this happens at a certin time
within a packet.
       RX_GOOD_PKT_TIME <= CONV_INTEGER(OUT_POSCTR);  --
record where the transmitter is at when a packet is
received and is good.
       GOOD_POSITION_LOCKOUT := '1';
     else
       --reset counter
       GOOD_POSITION_LOCKOUT := '0';
     end if;
    end if;
  end if;
end process;
end Behavioral;
```

9.2.20 Watchdog.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity WATCHDOG is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  WDT_EXPIRE: out std_logic;
  R: in std_logic;        --reset (edge sensitive,
synchronous)
  C: in std_logic;        --count (edge sensitive,
synchronous)

  WDT_THRESHOLD: in std_logic_vector( 7 downto 0 )
);
end WATCHDOG;

--Edge triggered watchdog timer.
architecture Behavioral of WATCHDOG is
  type T_FSM_WATCHDOG is ( IDLE, RESET, COUNT );
  signal STATE, NEXTSTATE : T_FSM_WATCHDOG;


  signal Rp, Cp: std_logic;

  signal WDT_SAT: std_logic;
  signal WDT_CTR: std_logic_vector( 7 downto 0 );
  signal WDT_THRESHOLD_BUF: std_logic_vector( 7 downto 0
);
begin

WDT_EXPIRE <= WDT_SAT;

--Shift registers to track time history of R (Reset) and
C (Clear) signals
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     Rp <= '1';
     Cp <= '1';
    else
     Rp <= R;
     Cp <= C;
    end if;
  end if;
end process;

--WDT counter management
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    WDT_THRESHOLD_BUF <= WDT_THRESHOLD;

    --determine saturation threshold
    --Can pipeline this because it is impossible to count
twice in consecutive cycles
    if( WDT_CTR >= WDT_THRESHOLD_BUF ) then
     WDT_SAT <= '1';
    else
     WDT_SAT <= '0';
    end if;

    --control counter - count or reset
    if( STATE = RESET or BLOCK_RESET_L = '0' ) then
     WDT_CTR <= "00000000";
    elsif( STATE = COUNT and WDT_SAT = '0') then
     WDT_CTR <= WDT_CTR + 1;
    end if;
  end if;

end process;

process( BLOCK_HSCLK ) is
begin
```

```
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXTSTATE;
    end if;
  end if;
end process;

process( STATE, R, C, Rp, Cp ) is
begin
  case( STATE ) is
  when IDLE =>
    --If rising edge on reset, go to reset state
    if( R = '1' and Rp = '0' ) then
     NEXTSTATE <= RESET;
    --If rising edge on count, go to count state
    elsif( C = '1' and Cp = '0' ) then
     NEXTSTATE <= COUNT;
    else
     NEXTSTATE <= IDLE;
    end if;
  when RESET =>
    --If rising edge on count, go to count state
    if( C = '1' and Cp = '0' ) then
     NEXTSTATE <= COUNT;
    else
     --Return to idle
     NEXTSTATE <= IDLE;
    end if;
  when COUNT =>
    --If rising edge on reset, then to go reset state
    if( R = '1' and Rp = '0' ) then
     NEXTSTATE <= RESET;
    else
     --Return to idle
     NEXTSTATE <= IDLE;
    end if;
  end case;
end process;

end Behavioral;
```

9.2.21 Netclk_Mgr.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity NETCLK_MGR is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  SETCLK: in std_logic;
  SETVAL: in T_NETTIME;

  INCREMENT: in std_logic;

  CURRTIME: out T_NETTIME;
  NEXTTIME: out T_NETTIME
);
end NETCLK_MGR;

architecture Behavioral of NETCLK_MGR is
signal NETCLK: T_NETTIME;
signal INCREMENT_LOCKOUT: std_logic;

begin
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    CURRTIME <= NETCLK;
    NEXTTIME <= NETCLK + 1;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     NETCLK <= X"00";
     INCREMENT_LOCKOUT <= '0';
    elsif( SETCLK = '1' ) then
     NETCLK <= SETVAL;
    elsif( INCREMENT = '1' and INCREMENT_LOCKOUT = '0' )
then
     NETCLK <= NETCLK + 1;
     INCREMENT_LOCKOUT <= '1';
    elsif( INCREMENT = '0' ) then
     INCREMENT_LOCKOUT <= '0';
    end if;
  end if;
end process;

end Behavioral;
```

9.2.22 Normal_DM.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity NORMAL_DM is
    Port (
    BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;
    BLOCK_PKT_GOOD: in std_logic;
    BLOCK_PKT_RDY: out std_logic;
    BLOCK_PKT_CLR: in std_logic;

    BLOCK_CMD_IN: in T_COMMAND;
    BLOCK_CMD_OUT: out T_COMMAND;
    NET_TIME: in T_NETTIME;

    BLOCK_DATA_IN: in T_DATA_PAYLOAD;
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;
    BLOCK_FROM_ADDR_IN: in T_NODEADDR;
    BLOCK_TO_ADDR_OUT: out T_NODEADDR;

    ACTIVE_DATA: out T_NORMAL_DATA_ARRAY;

    PKT_MODE: in E_PACKET_MODE;

    --Status/Sensor Information
    STATUS_DATA: in T_NORMAL_DATA_ARRAY;

    SYNC: out std_logic
    );
end NORMAL_DM;

architecture Behavioral of NORMAL_DM is
  type T_NORMAL_FSM is ( idle, ready, waitclr );
  signal STATE, NEXT_STATE: T_NORMAL_FSM;

  component PIPE_COMPARE is
  generic(
    VECSIZE: integer; RST_VAL: std_logic := '0');
  port(
    RESET_L: in std_logic;
    CLK: in std_logic;
    VALUE: in std_logic_vector( VECSIZE - 1 downto 0 );
    TARGET: in std_logic_vector( VECSIZE - 1 downto 0 );
    RESULT: out std_logic);
  end component;

  --scheduled, but inactive data
  signal SCHED_DATA: T_NORMAL_DATA_ARRAY;
  signal SCHED_TIME: T_NETTIME;

  --active data temporary signal
  signal ACTIVE_DATAX: T_NORMAL_DATA_ARRAY;
  signal ENABLE_XFER: std_logic;

  --command normal indicator
  signal CMD_NORMAL: std_logic;
begin
--move dummy signal to real signal
ACTIVE_DATA <= ACTIVE_DATAX;


--Move updates when they are scheduled
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( ENABLE_XFER = '1' ) then
     ACTIVE_DATAX <= SCHED_DATA;
    end if;
  end if;
```

```
end process;

--Using an enable signal increased clock frequency
instead of directly comaring in above process
process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( SCHED_TIME = NET_TIME ) then
     ENABLE_XFER <= '1';
     SYNC    <= '1';
    else
     ENABLE_XFER <= '0';
     SYNC <= '0';
    end if;
  end if;
end process;

--structure of normal packet data by byte is:
--TIME, DIR, ATTR1(15 downto 8), ATTR1(7 downto 0),
ATTR2(15 downto 8), ATTR2(7 downto 0),
-- ATTR3(15 downto 8), ATTR3(7 downto 0), ATTR4(15
downto 8), ATTR4(7 downto 0)
process( BLOCK_RESET_L, BLOCK_CMD_IN,  BLOCK_HSCLK,
BLOCK_PKT_COMPLETE ) is
variable LOCKOUT: std_logic;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     SCHED_TIME <= "00000000";
     SCHED_DATA(0) <= "0000000000000000";
     SCHED_DATA(1) <= "0000000000000000";
     SCHED_DATA(2) <= "0000000000000000";
     SCHED_DATA(3) <= "0000000000000000";
     LOCKOUT := '0';
    else
     --in this block BLOCK_PKT_GOOD also checks ADDR_THIS
     --Sample data when packet is of type normal, and
lockout future samples until
     --a new packet arrives
     if( BLOCK_PKT_GOOD = '1' and CMD_NORMAL = '1' and
LOCKOUT = '0' ) then
       LOCKOUT := '1';
       --Store new values if DIR = '0'
       if( BLOCK_DATA_IN(1)(0) = '0' ) then
         --ATTR 1
         SCHED_DATA(0) <= BLOCK_DATA_IN(2) &
BLOCK_DATA_IN(3);
         --SCHEDULED TIME
         SCHED_TIME <= BLOCK_DATA_IN(0);
       end if;

       if( BLOCK_DATA_IN(1)(0) = '0' ) then
         --ATTR 2
         SCHED_DATA(1) <= BLOCK_DATA_IN(4) &
BLOCK_DATA_IN(5);
         --ATTR 3
         SCHED_DATA(2) <= BLOCK_DATA_IN(6) &
BLOCK_DATA_IN(7);
         --ATTR 4
         SCHED_DATA(3) <= BLOCK_DATA_IN(8) &
BLOCK_DATA_IN(9);
       end if;
     --Reset lockout after BLOCK_PKT_GOOD falls
     elsif( BLOCK_PKT_GOOD = '0' ) then
       LOCKOUT := '0';
     end if;
    end if;
  end if;
end process;

PIPE_COMPARE1: PIPE_COMPARE generic map (VECSIZE => 4,
RST_VAL => '0')
   port map( RESET_L => BLOCK_RESET_L, CLK =>
BLOCK_HSCLK, VALUE => BLOCK_CMD_IN,
       TARGET => C_CMD_NORMAL, RESULT => CMD_NORMAL );
--send command back out
BLOCK_CMD_OUT <= C_CMD_NORMAL;

--structure of normal packet data by byte is:
--TIME, DIR, ATTR1(15 downto 8), ATTR1(7 downto 0),
ATTR2(15 downto 8), ATTR2(7 downto 0),
-- ATTR3(15 downto 8), ATTR3(7 downto 0), ATTR4(15
downto 8), ATTR4(7 downto 0)
process( BLOCK_CMD_IN,  BLOCK_HSCLK ) is
```

```
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( CMD_NORMAL = '1' ) then
     --must be a packet of type command normal
(otherwise, it is not for this manager to deal with)
    if( BLOCK_PKT_GOOD = '1' and BLOCK_DATA_IN(1)(0) =
'0' ) then
       BLOCK_TO_ADDR_OUT <= BLOCK_FROM_ADDR_IN;
       BLOCK_DATA_OUT(0) <= BLOCK_DATA_IN(0);
  --Copy TIME back so DSP can identify packet
       BLOCK_DATA_OUT(1) <= "00000001";
  --Make DIR=1 because it is going out
       BLOCK_DATA_OUT(2) <= STATUS_DATA(0)(15 downto 8);
  --Put Status Data in field
       BLOCK_DATA_OUT(3) <= STATUS_DATA(0)( 7 downto 0);
  --Put Status Data in field
       BLOCK_DATA_OUT(4) <= STATUS_DATA(1)(15 downto 8);
--Put Status Data in field
       BLOCK_DATA_OUT(5) <= STATUS_DATA(1)( 7 downto 0);
  --Put Status Data in field
       BLOCK_DATA_OUT(6) <= STATUS_DATA(2)(15 downto 8);
  --Put Status Data in field
       BLOCK_DATA_OUT(7) <= STATUS_DATA(2)( 7 downto 0);
  --Put Status Data in field
       BLOCK_DATA_OUT(8) <= STATUS_DATA(3)(15 downto 8);
  --Put Status Data in field
       BLOCK_DATA_OUT(9) <= STATUS_DATA(3)( 7 downto 0);
  --Put Status Data in field
  end if;
    end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXT_STATE;
    end if;
  end if;
end process;

process( STATE, BLOCK_PKT_GOOD, BLOCK_DATA_IN,
CMD_NORMAL, BLOCK_PKT_CLR ) is
begin
  case( STATE ) is
  when idle =>
    BLOCK_PKT_RDY <= '0';
    --wait for a normal packet that is destined for this
node
    if( BLOCK_PKT_GOOD = '1' and BLOCK_DATA_IN(1)(0) =
'0' and CMD_NORMAL = '1') then
     NEXT_STATE <= ready;
    else
     NEXT_STATE <= idle;
    end if;
  when ready =>
    BLOCK_PKT_RDY <= '1';
    --wait for clr to go high, indicating packet is
absorbed
    if( BLOCK_PKT_CLR = '1' ) then
     NEXT_STATE <= waitclr;
    else
     NEXT_STATE <= ready;
    end if;
  when waitclr =>
    BLOCK_PKT_RDY <= '0';
    --wait for clear to fall back to 0 and for the packet
to end
    --wait for block_pkt_clr to fall to prevent re-entry
    if( BLOCK_PKT_CLR = '0' and BLOCK_PKT_GOOD = '0')
then
     NEXT_STATE <= idle;
    else
     NEXT_STATE <= waitclr;
    end if;
  when others =>
    BLOCK_PKT_RDY <= '0';
    NEXT_STATE <= idle;
  end case;
end process;
end Behavioral;
```

9.2.23 Redirector.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the
declarations that are
--  provided for instantiating Xilinx primitive
components.
--library UNISIM;
--use UNISIM.VComponents.all;
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity redirector is
  port(
     --from top layer
     REDIR_OUT_CMD: out T_COMMAND;
     REDIR_OUT_ADDR_TO:  out T_NODEADDR;
     REDIR_OUT_ADDR_FROM: out T_NODEADDR;
     REDIR_OUT_DATA: out T_DATA_PAYLOAD;

     REDIR_IN_CMD: in T_COMMAND;
     REDIR_IN_ADDR_TO:  in T_NODEADDR;
     REDIR_IN_ADDR_FROM: in T_NODEADDR;
     REDIR_IN_DATA: in T_DATA_PAYLOAD;

     --From bottom layer
     REDIR_OUT1_CMD: out T_COMMAND;
     REDIR_OUT1_ADDR_TO:  out T_NODEADDR;
     REDIR_OUT1_ADDR_FROM: out T_NODEADDR;
     REDIR_OUT1_DATA: out T_DATA_PAYLOAD;

     REDIR_IN1_CMD: in T_COMMAND;
     REDIR_IN1_ADDR_TO:  in T_NODEADDR;
     REDIR_IN1_ADDR_FROM: in T_NODEADDR;
     REDIR_IN1_DATA: in T_DATA_PAYLOAD);

end redirector;

architecture Behavioral of redirector is
begin

--just forward the data for now and don't use redirector

--Data going down to transmitter
REDIR_OUT1_CMD <= REDIR_IN_CMD;
REDIR_OUT1_ADDR_TO <= REDIR_IN_ADDR_TO;
REDIR_OUT1_ADDR_FROM <= REDIR_IN_ADDR_FROM;
REDIR_OUT1_DATA <= REDIR_IN_DATA;

--Data going up to command processor
REDIR_OUT_CMD <= REDIR_IN1_CMD;
REDIR_OUT_ADDR_TO <= REDIR_IN1_ADDR_TO;
REDIR_OUT_ADDR_FROM <= REDIR_IN1_ADDR_FROM;
REDIR_OUT_DATA <= REDIR_IN1_DATA;

end Behavioral;
```

9.2.24 Sched_DM.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Use PEBB PNP library for communications
library COMMLIB;
use COMMLIB.all;
use COMMLIB.COMMSTD.all;

entity SCHED_DM is
    Port (
    BLOCK_HSCLK: in std_logic;  --clock input
    BLOCK_RESET_L: in std_logic;  --reset signal

    BLOCK_PKT_COMPLETE: in std_logic;
    BLOCK_PKT_GOOD: in std_logic;

    BLOCK_PKT_RDY: out std_logic;
    BLOCK_PKT_CLR: in std_logic;

    BLOCK_CMD: in T_COMMAND;
    NET_TIME: in T_NETTIME;

    BLOCK_CMD_OUT: out T_COMMAND;
    BLOCK_DATA_IN: in T_DATA_PAYLOAD;
    BLOCK_DATA_OUT: out T_DATA_PAYLOAD;
    BLOCK_FROM_ADDR_IN: in T_NODEADDR;
    BLOCK_TO_ADDR_OUT: out T_NODEADDR;

    ACTIVE_DATA: out T_SYNC_ATTR_ARRAY;

    PKT_MODE: in E_PACKET_MODE
   );
end SCHED_DM;

architecture Behavioral of SCHED_DM is
  type T_FSM_SCHED is (idle, ready, waitclr );
  signal STATE, NEXT_STATE: T_FSM_SCHED;

  --scheduled, but inactive data
  signal SCHED_DATA: T_SYNC_ATTR_ARRAY;

  type T_SCHED_TIME_ARRAY is array( C_ACTIVE_DATA_MAX
downto 0 ) of T_NETTIME;
  signal SCHED_TIME: T_SCHED_TIME_ARRAY;

  signal SCHED_TIME_TRANSFER: std_logic_vector(
C_ACTIVE_DATA_MAX downto 0 );

  --active data temporary signal
  signal ACTIVE_DATAX: T_SYNC_ATTR_ARRAY;

  --address of attributes to read
  signal ATTR1: std_logic_vector( 4 downto 0 );
  signal ATTR2: std_logic_vector( 4 downto 0 );

  --Values read from syncnronous attributes
  signal RD_ATTR1: T_DATA;
  signal RD_ATTR2: T_DATA;

  signal WR_EN1: std_logic_vector( C_ACTIVE_DATA_MAX
downto 0 );
  signal SETVAL_SYNC_EN: std_logic;

  signal CMD_GETSYNC: std_logic;
begin
--move dummy signal to real signal
ACTIVE_DATA <= ACTIVE_DATAX;

--Move updates when they are scheduled
G1:  for i in 0 to 7 generate
     process( BLOCK_RESET_L, BLOCK_HSCLK ) is
     begin
       --if( BLOCK_RESET_L = '0' ) then
       --  SCHED_TIME_TRANSFER(i) <= '0';
       if( rising_edge( BLOCK_HSCLK ) ) then
```

```
        if( BLOCK_RESET_L = '0' ) then
          SCHED_TIME_TRANSFER(i) <= '0';
        else
          if( SCHED_TIME(i) = NET_TIME ) then
           SCHED_TIME_TRANSFER(i) <= '1';
          else
           SCHED_TIME_TRANSFER(i) <= '0';
          end if;
        end if;
      end if;
    end process;

    process( BLOCK_RESET_L, BLOCK_HSCLK ) is
    begin
      if( rising_edge( BLOCK_HSCLK ) ) then
        if( BLOCK_RESET_L = '0' ) then
          ACTIVE_DATAX(i) <= X"0000";
        else
          if( SCHED_TIME_TRANSFER(i) = '1' ) then
           ACTIVE_DATAX(i) <= SCHED_DATA(i);
          end if;
        end if;
      end if;
    end process;

    process( BLOCK_HSCLK, BLOCK_RESET_L ) is
    begin
      --set initial conditions to zero
      if( rising_edge( BLOCK_HSCLK ) ) then
      if(BLOCK_RESET_L = '0') then
        SCHED_DATA(i) <= "0000000000000000";
        SCHED_TIME(i) <= "00000000";
        WR_EN1(i) <= '0';
      else

        --If the attribute destination is the same as
this node's attribute enable write strobe for this node
        if( ATTR1(4) = '1' and ATTR1(3 downto 0) =
CONV_STD_LOGIC_VECTOR( i, 4) ) then
          WR_EN1(i) <= '1';
        else
          WR_EN1(i) <= '0';
        end if;

        --if write strobe is enabled and packet is
valid, then copy data
        if( WR_EN1(i) = '1' and BLOCK_PKT_GOOD = '1' and
SETVAL_SYNC_EN = '1') then
          SCHED_DATA(i) <= BLOCK_DATA_IN(2) &
BLOCK_DATA_IN(3);

          --Store new time to active attribute 1
          SCHED_TIME(i) <= BLOCK_DATA_IN(0);
        end if;
       end if;
       end if;
     end process;
    end generate;

G2: for i in 8 to 15 generate
  SCHED_DATA(i) <= X"0000";
  SCHED_TIME(i) <= X"00";
  ACTIVE_DATAX(i) <= X"0000";
  WR_EN1(i) <= '0';
end generate;

--Assign temporary signal for addresses
ATTR1 <= BLOCK_DATA_IN(1)(4 downto 0);
ATTR2 <= BLOCK_DATA_IN(5)(4 downto 0);

--raise the maximum clock frequency by pipelining the
compare
process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     SETVAL_SYNC_EN <= '0';
    else
     if( BLOCK_CMD = C_CMD_SETVAL_SYNC ) then
       SETVAL_SYNC_EN <= '1';
     else
       SETVAL_SYNC_EN <= '0';
     end if;
    end if;
```

```
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
     CMD_GETSYNC <= '0';
    else
     if( BLOCK_CMD = C_CMD_GETVAL_SYNC ) then
       CMD_GETSYNC <= '1';
     else
       CMD_GETSYNC <= '0';
     end if;
    end if;
  end if;
end process;

--Allways decode the data so that it is ready for
transmission when the command is recognized.
process( BLOCK_HSCLK ) is
  variable RD_ATTR1A: T_DATA;
  variable RD_ATTR1B: T_DATA;
  variable RD_ATTR1C: T_DATA;
  variable RD_ATTR1D: T_DATA;
  variable RD_ATTR2A: T_DATA;
  variable RD_ATTR2B: T_DATA;
  variable RD_ATTR2C: T_DATA;
  variable RD_ATTR2D: T_DATA;
begin
  if( rising_edge( BLOCK_HSCLK ) ) then

  --try a pipelined version for increased speed
     case ATTR1(3 downto 2) is
       when "00" => RD_ATTR1 <= RD_ATTR1A;
       when "01" => RD_ATTR1 <= RD_ATTR1B;
       when "10" => RD_ATTR1 <= RD_ATTR1C;
       when "11" => RD_ATtR1 <= RD_ATTR1D;
       when others => null;
     end case;

     case ATTR2(3 downto 2) is
       when "00" => RD_ATTR2 <= RD_ATTR2A;
       when "01" => RD_ATTR2 <= RD_ATTR2B;
       when "10" => RD_ATTR2 <= RD_ATTR2C;
       when "11" => RD_ATtR2 <= RD_ATTR2D;
       when others => null;
     end case;

     RD_ATTR1A := ACTIVE_DATAX( CONV_INTEGER( "00" &
ATTR1(1 downto 0 )));
     RD_ATTR1B := ACTIVE_DATAX( CONV_INTEGER( "01" &
ATTR1(1 downto 0 )));
     RD_ATTR1C := ACTIVE_DATAX( CONV_INTEGER( "10" &
ATTR1(1 downto 0 )));
     RD_ATTR1D := ACTIVE_DATAX( CONV_INTEGER( "11" &
ATTR1(1 downto 0 )));

     RD_ATTR2A := ACTIVE_DATAX( CONV_INTEGER( "00" &
ATTR2(1 downto 0 )));
     RD_ATTR2B := ACTIVE_DATAX( CONV_INTEGER( "01" &
ATTR2(1 downto 0 )));
     RD_ATTR2C := ACTIVE_DATAX( CONV_INTEGER( "10" &
ATTR2(1 downto 0 )));
     RD_ATTR2D := ACTIVE_DATAX( CONV_INTEGER( "11" &
ATTR2(1 downto 0 )));
  end if;
end process;

BLOCK_CMD_OUT <= C_CMD_GETVAL_SYNC;

--structure of sync packet data is: TIME, ATTR, VALUE,
TIME, ATTR, VALUE
process( BLOCK_CMD,  BLOCK_HSCLK ) is
begin
     --hold data constant when not in idle state
    if( STATE = idle ) then
     BLOCK_DATA_OUT(0) <= X"01";
    --Copy same back so DSP can identify packet
     BLOCK_DATA_OUT(1) <= BLOCK_DATA_IN(1);
  --Copy same back so DSP can identify packet
     BLOCK_DATA_OUT(2) <= RD_ATTR1(15 downto 8);      --
Put contents of active value of attribute in field
```

```
    BLOCK_DATA_OUT(3) <= RD_ATTR1( 7 downto 0);      --
Put contents of active value of attribute in field
    BLOCK_DATA_OUT(4) <= BLOCK_DATA_IN(4);           --
Copy same back so DSP can identify packet
    BLOCK_DATA_OUT(5) <= BLOCK_DATA_IN(5);
  --Copy same back so DSP can identify packet
    BLOCK_DATA_OUT(6) <= RD_ATTR2(15 downto 8);      --
Put contents of active value of attribute in field
    BLOCK_DATA_OUT(7) <= RD_ATTR2( 7 downto 0);      --
Put contents of active value of attribute in field
    BLOCK_DATA_OUT(8) <= BLOCK_DATA_IN(8);
  --Copy same back so DSP can identify packet
    BLOCK_DATA_OUT(9) <= BLOCK_DATA_IN(9);
  --Copy same back so DSP can identify packet

    BLOCK_TO_ADDR_OUT <= BLOCK_FROM_ADDR_IN;
   end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
   if( BLOCK_RESET_L = '0' ) then
    STATE <= idle;
   else
    STATE <= NEXT_STATE;
   end if;
  end if;
end process;

process( STATE, BLOCK_PKT_GOOD, BLOCK_DATA_IN,
CMD_GETSYNC, BLOCK_PKT_CLR ) is
begin
  case( STATE ) is
  when idle =>
   BLOCK_PKT_RDY <= '0';
   --wait for a normal packet that is destined for this
node
   if( BLOCK_PKT_GOOD = '1' and CMD_GETSYNC = '1' and
BLOCK_DATA_IN(0)(0) = '0') then
    NEXT_STATE <= ready;
   else
    NEXT_STATE <= idle;
   end if;
  when ready =>
   BLOCK_PKT_RDY <= '1';
   --wait for clr to go high, indicating packet is
absorbed
   if( BLOCK_PKT_CLR = '1' ) then
    NEXT_STATE <= waitclr;
   else
    NEXT_STATE <= ready;
   end if;
  when waitclr =>
   BLOCK_PKT_RDY <= '0';
   --wait for clear to fall back to 0 and for the packet
to end
   --wait for block_pkt_clr to fall to prevent re-entry
   if( BLOCK_PKT_CLR = '0' and BLOCK_PKT_GOOD = '0')
then
    NEXT_STATE <= idle;
   else
    NEXT_STATE <= waitclr;
   end if;
  when others =>
   BLOCK_PKT_RDY <= '0';
   NEXT_STATE <= idle;
  end case;
end process;

end Behavioral;
```

9.2.25 ControlReg.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity controlreg is
    Port (
   BLOCK_HSCLK: in std_logic;
   BLOCK_DONE: out std_logic;
   BLOCK_RESET_L: in stD_logic;
   BLOCK_ENABLE: in std_logic;
   BLOCK_DATA_IN: in std_logic_vector( 31 downto 0 );
   BLOCK_DATA_OUT: out std_logic_vector( 31 downto 0 );
   BLOCK_ADDR_IN: in std_logic_vector( 19 downto 0 );
   BLOCK_WRITE_L: in std_logic;

   -- DAC control/status signals
   DACMSB: out std_logic;
   DACRST: out std_logic;

   -- DSP control/status signals
   DSPSTAT: in std_logic_vector( 7 downto 0 );

   --Fiber optic control/status signals
   FO_CONTROL: out std_logic_vector( 1 downto 0 );
   FO_STATUS: in std_logic_vector( 7 downto 0 );

   FO2_FIBER_ADDR: out std_logic_vector( 15 downto 0 );

   FO2_PHASEA: out std_logic_vector( 31 downto 0 );
   FO2_PHASEB: out std_logic_vector( 31 downto 0 );
   FO2_PHASEC: out std_logic_vector( 31 downto 0 );

   FO2_SENSORA: in std_logic_vector( 31 downto 0 );
   FO2_SENSORB: in std_logic_vector( 31 downto 0 );
   FO2_SENSORC: in std_logic_vector( 31 downto 0 );

   --Hex display control/status signals
   HEXBLANK: out std_logic;
   UNUSED: in std_logic_vector( 31 downto 0 )
  );
end controlreg;

architecture Behavioral of controlreg is
  --from selector.vhd
  constant ADDR_DAC:     std_logic_vector( 3 downto 0 )
:=  "0001";
  constant ADDR_DIPSW:   std_logic_vector( 3 downto 0 )
:=  "0010";
  constant ADDR_DSP:     std_logic_vector( 3 downto 0 )
:=  "1011";
  constant ADDR_FPGA:    std_logic_vector( 3 downto 0 )
:=  "0100";
  constant ADDR_HEX:     std_logic_vector( 3 downto 0 )
:=  "0101";
  constant ADDR_PERIPH:  std_logic_vector( 3 downto 0 )
:=  "0110";
  constant ADDR_PEEPROM: std_logic_vector( 3 downto 0 )
:=  "0111";
  constant ADDR_PESNET:  std_logic_vector( 3 downto 0 )
:=  "1000";
  constant ADDR_PMC:     std_logic_vector( 3 downto 0 )
:=  "1001";
  constant ADDR_ULCTL:   std_logic_vector( 3 downto 0 )
:=  "1010";
  constant ADDR_DEBUG:   std_logic_vector( 3 downto 0 )
:=  "1111";

  type FSM_STATE is (idle, preactive, active, done);
  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;

  --HEX Control Register
  signal REG_HEXBLANK: std_logic;
  constant INIT_HEXBLANK: std_logic := '0';
--  constant ADDR_HEXBLANK: std_logic_vector( 7 downto 0
) := ADDR_HEX & "0000";
```

```
  constant ADDR_HEXBLANK: std_logic_vector( 7 downto 0 )
:= "01010000";

  --DAC Control register
  --Bit 0 = DACRST
  --Bit 1 = DACMSB
  signal REG_DACCTL: std_logic_vector( 1 downto 0 );
  constant INIT_DACCTL: std_logic_vector( 1 downto 0 )
:= "00";
--  constant ADDR_DACCTL: std_logic_vector( 7 downto 0 )
:= ADDR_DAC & "0000";
  constant ADDR_DACCTL: std_logic_vector( 7 downto 0 )
:= "00010000";
  constant CN_DACRST: integer := 0;  -- bit position of
DACRST
  constant CN_DACMSB: integer := 1;  -- bit position of
DACMSB

  signal REG_DSPSTAT: std_logic_vector( 7 downto 0 );
--  constant ADDR_DSPSTAT: std_logic_vector( 7 downto 0 )
:= ADDR_DSP & "0000";
    constant ADDR_DSPSTAT: std_logic_vector( 7 downto 0 )
:= "10110000";

--  constant ADDR_VERSION: std_logic_vector( 7 downto 0 )
:= ADDR_FPGA & "1011";
  constant ADDR_VERSION: std_logic_vector( 7 downto 0 )
:= "01001011";

  constant REG_VERSION: std_logic_vector( 15 downto 0 )
:= CONV_STD_LOGIC_VECTOR(16,16);


  --a simple register for storage / test
  signal REG_FEEDBACK: std_logic_vector( 31 downto 0 );
--  constant ADDR_FEEDBACK: std_logic_vector( 7 downto 0
) := ADDR_FPGA & "0000";
  constant ADDR_FEEDBACK: std_logic_vector( 7 downto 0 )
:= "01000000";
  constant INIT_FEEDBACK: std_logic_vector( 31 downto 0
) := "00000000000000000000000000000000";

  signal REG_UNUSED: std_logic_vector( 31 downto 0 );
--  constant ADDR_UNUSED: std_logic_vector( 7 downto 0 )
:= ADDR_FPGA & "1000";
  constant ADDR_UNUSED: std_logic_vector( 7 downto 0 )
:= "01001000";

  signal REG_FO_CONTROL: std_logic_vector( 1 downto 0 );
--  constant ADDR_FO_CONTROL: std_logic_vector( 7 downto
0 ) :=  ADDR_PESNET & "0000";
  constant ADDR_FO_CONTROL: std_logic_vector( 7 downto 0
) :=   "10000000";
  constant INIT_FO_CONTROL: std_logic_vector( 1 downto 0
) := "00";

  signal REG_FO_STATUS: std_logic_vector( 7 downto 0 );
--  constant ADDR_FO_STATUS: std_logic_vector( 7 downto 0
) :=   ADDR_PESNET & "0001";
    constant ADDR_FO_STATUS: std_logic_vector( 7 downto 0
) :=   "10000001";

  signal REG_FO2_PHASEA: std_logic_vector( 31 downto 0
);
--  constant ADDR_FO2_PHASEA: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "0100";
  constant ADDR_FO2_PHASEA: std_logic_vector( 7 downto 0
) := "10000100";
  constant INIT_FO2_PHASEA: std_logic_vector( 31 downto
0 ) := "00000000000000000000000000000000";

  signal REG_FO2_PHASEB: std_logic_vector( 31 downto 0
);
--  constant ADDR_FO2_PHASEB: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "0101";
  constant ADDR_FO2_PHASEB: std_logic_vector( 7 downto 0
) := "10000101";
  constant INIT_FO2_PHASEB: std_logic_vector( 31 downto
0 ) := "00000000000000000000000000000000";

  signal REG_FO2_PHASEC: std_logic_vector( 31 downto 0
);
--  constant ADDR_FO2_PHASEC: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "0110";
```

```
  constant ADDR_FO2_PHASEC: std_logic_vector( 7 downto 0
) := "10000110";
  constant INIT_FO2_PHASEC: std_logic_vector( 31 downto
0 ) := "00000000000000000000000000000000";

  signal REG_FO2_SENSORA: std_logic_vector( 31 downto 0
);
-- constant ADDR_FO2_SENSORA: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "1000";
  constant ADDR_FO2_SENSORA: std_logic_vector( 7 downto
0 ) := "10001000";

  signal REG_FO2_SENSORB: std_logic_vector( 31 downto 0
);
-- constant ADDR_FO2_SENSORB: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "1001";
  constant ADDR_FO2_SENSORB: std_logic_vector( 7 downto
0 ) := "10001001";

  signal REG_FO2_SENSORC: std_logic_vector( 31 downto 0
);
-- constant ADDR_FO2_SENSORC: std_logic_vector( 7 downto
0 ) := ADDR_PESNET & "1010";
  constant ADDR_FO2_SENSORC: std_logic_vector( 7 downto
0 ) := "10001010";

  signal REG_FO2_FIBER_ADDR: std_logic_vector(15 downto
0 );
-- constant ADDR_FO2_FIBER_ADDR: std_logic_vector( 7
downto 0 ) := ADDR_PESNET & "1011";
  constant ADDR_FO2_FIBER_ADDR: std_logic_vector( 7
downto 0 ) := "10001011";
  constant INIT_FO2_FIBER_ADDR: std_logic_vector( 15
downto 0 ) := "0110001000110111";  --0x6327

  signal COUNTER: integer range 0 to 31;

  signal WR_HEXBLANK: std_logic;
  signal WR_DACCTL: std_logic;
  signal WR_FEEDBACK: std_logic;
  signal WR_FO_CONTROL: std_logic;
  signal WR_FO2_FIBERADDR: std_logic;
  signal WR_FO2_PHASEA: std_logic;
  signal WR_FO2_PHASEB: std_logic;
  signal WR_FO2_PHASEC: std_logic;

  signal LOOKUP_PESNET: std_logic_vector( 31 downto 0 );
  signal LOOKUP_OTHERS: std_logic_vector( 31 downto 0 );

begin

--D to A signals
DACMSB <= REG_DACCTL(CN_DACMSB);
DACRST <= REG_DACCTL(CN_DACRST);


FO_CONTROL <= REG_FO_CONTROL;

FO2_PHASEA <= REG_FO2_PHASEA;
FO2_PHASEB <= REG_FO2_PHASEB;
FO2_PHASEC <= REG_FO2_PHASEC;
FO2_FIBER_ADDR <= REG_FO2_FIBER_ADDR;

--Hex Display signals
HEXBLANK <= REG_HEXBLANK;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    REG_FO_STATUS <= FO_STATUS;
    REG_FO2_SENSORA <= FO2_SENSORA;
    REG_FO2_SENSORB <= FO2_SENSORB;
    REG_FO2_SENSORC <= FO2_SENSORC;
    REG_UNUSED <= UNUSED;
    REG_DSPSTAT <= DSPSTAT;
  end if;
end process;


--manage state
process( BLOCK_RESET_L, BLOCK_ENABLE, BLOCK_HSCLK )
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0' ) then
```

```
    STATE <= idle;
    else
      STATE <= NEXTSTATE;
   end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
   if( BLOCK_RESET_L = '0' ) then
    LOOKUP_PESNET <= X"00000000";
   else
    case BLOCK_ADDR_IN(8 downto 1 ) is
      when ADDR_FO_CONTROL =>    LOOKUP_PESNET <=
CONV_STD_LOGIC_VECTOR(0, 30)  & REG_FO_CONTROL;
      when ADDR_FO_STATUS =>           LOOKUP_PESNET <=
CONV_STD_LOGIC_VECTOR(0, 24)  & REG_FO_STATUS;
      when ADDR_FO2_FIBER_ADDR =>      LOOKUP_PESNET <=
CONV_STD_LOGIC_VECTOR(0, 16)  & REG_FO2_FIBER_ADDR;
      when ADDR_FO2_PHASEA =>          LOOKUP_PESNET <=
REG_FO2_PHASEA;
      when ADDR_FO2_PHASEB =>          LOOKUP_PESNET <=
REG_FO2_PHASEB;
      when ADDR_FO2_PHASEC =>          LOOKUP_PESNET <=
REG_FO2_PHASEC;
      when ADDR_FO2_SENSORA =>         LOOKUP_PESNET <=
REG_FO2_SENSORA;
      when ADDR_FO2_SENSORB =>         LOOKUP_PESNET <=
REG_FO2_SENSORB;
      when ADDR_FO2_SENSORC =>         LOOKUP_PESNET <=
REG_FO2_SENSORC;
      when others => LOOKUP_PESNET <=  LOOKUP_OTHERS;
    end case;
   end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
   if( BLOCK_RESET_L = '0' ) then
    LOOKUP_OTHERS <= X"00000000";
   else
    case BLOCK_ADDR_IN(8 downto 1 ) is
      when ADDR_DACCTL =>              LOOKUP_OTHERS <=
CONV_STD_LOGIC_VECTOR(0, 30 ) & REG_DACCTL;
      when ADDR_DSPSTAT =>             LOOKUP_OTHERS <=
CONV_STD_LOGIC_VECTOR(0, 24 ) & REG_DSPSTAT;
      when ADDR_FEEDBACK =>            LOOKUP_OTHERS <=
REG_FEEDBACK;
      when ADDR_VERSION =>             LOOKUP_OTHERS <=
CONV_STD_LOGIC_VECTOR(0, 16) &  REG_VERSION;
      when ADDR_HEXBLANK =>            LOOKUP_OTHERS <=
CONV_STD_LOGIC_VECTOR(0, 31)  & REG_HEXBLANK;
      when ADDR_UNUSED =>              LOOKUP_OTHERS <=
REG_UNUSED;
   --   when ADDR_FO2_SENSORA =>       LOOKUP_OTHERS <=
REG_FO2_SENSORA;
   --   when ADDR_FO2_SENSORB =>       LOOKUP_OTHERS <=
REG_FO2_SENSORB;
   --   when ADDR_FO2_SENSORC =>       LOOKUP_OTHERS <=
REG_FO2_SENSORC;
      when others =>                   LOOKUP_OTHERS <=
  X"DEEDDEED";
    end case;
   end if;
  end if;
end process;

BLOCK_DATA_OUT <= LOOKUP_PESNET;

process( STATE, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then

   if( STATE /= preactive ) then
    COUNTER <= 0;
   else
    COUNTER <= COUNTER + 1;
   end if;
  end if;
end process;
```

```
--Manage state
process( STATE, BLOCK_ADDR_IN, BLOCK_DATA_IN,
BLOCK_WRITE_L, COUNTER ) is
begin
  case( STATE ) is
    when idle =>
     NEXTSTATE <= preactive;
     BLOCK_DONE <= '1';
    when preactive =>
     if( COUNTER > 5 ) then
       NEXTSTATE <= active;
     else
       NEXTSTATE <= preactive;
     end if;
     BLOCK_DONE <= '0';
    when active =>
     NEXTSTATE <= done;
     BLOCK_DONE <= '0';
    when done =>
     NEXTSTATE <= done;
     BLOCK_DONE <= '1';
  end case;
end process;

process( BLOCK_RESET_L, BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_RESET_L = '0' ) then
       REG_DACCTL          <= INIT_DACCTL;
     REG_FEEDBACK       <= INIT_FEEDBACK;
     REG_FO_CONTROL     <= INIT_FO_CONTROL;
     REG_FO2_PHASEA     <= INIT_FO2_PHASEA;
     REG_FO2_PHASEB     <= INIT_FO2_PHASEB;
     REG_FO2_PHASEC     <= INIT_FO2_PHASEC;
     REG_FO2_FIBER_ADDR  <= INIT_FO2_FIBER_ADDR;
     REG_HEXBLANK       <= INIT_HEXBLANK;

     WR_HEXBLANK     <= '0';
     WR_DACCTL          <= '0';
     WR_FEEDBACK     <= '0';
     WR_FO_CONTROL      <= '0';
     WR_FO2_FIBERADDR   <= '0';
     WR_FO2_PHASEA      <= '0';
     WR_FO2_PHASEB      <= '0';
     WR_FO2_PHASEC      <= '0';

    else
     if( STATE = active ) then
       --manage writes
       if( BLOCK_WRITE_L = '0' ) then
         case BLOCK_ADDR_IN(8 downto 1 ) is
         when ADDR_HEXBLANK =>  WR_HEXBLANK       <= '1';
         when ADDR_DACCTL   =>  WR_DACCTL         <= '1';
         when ADDR_FEEDBACK =>  WR_FEEDBACK       <= '1';
         when ADDR_FO_CONTROL =>  WR_FO_CONTROL   <= '1';
         when ADDR_FO2_FIBER_ADDR =>  WR_FO2_FIBERADDR <=
'1';
         when ADDR_FO2_PHASEA =>  WR_FO2_PHASEA <= '1';
         when ADDR_FO2_PHASEB =>  WR_FO2_PHASEB <= '1';
         when ADDR_FO2_PHASEC =>  WR_FO2_PHASEC <= '1';
         when others => NULL;
         end case;
       end if;
     else
       WR_HEXBLANK      <= '0';
       WR_DACCTL        <= '0';
       WR_FEEDBACK      <= '0';
       WR_FO_CONTROL    <= '0';
       WR_FO2_FIBERADDR <= '0';
       WR_FO2_PHASEA    <= '0';
       WR_FO2_PHASEB    <= '0';
       WR_FO2_PHASEC    <= '0';
     end if;

     if( WR_HEXBLANK = '1' ) then
       REG_HEXBLANK <= BLOCK_DATA_IN( 0 );
     end if;

     if( WR_DACCTL = '1' ) then
       REG_DACCTL <= BLOCK_DATA_IN( 1 downto 0 );
     end if;

     if( WR_FEEDBACK = '1' ) then
       REG_FEEDBACK <= BLOCK_DATA_IN;
```

```
    end if;

    if( WR_FO_CONTROL = '1' ) then
      REG_FO_CONTROL <= BLOCK_DATA_IN(1 downto 0 );
    end if;

    if( WR_FO2_FIBERADDR = '1' ) then
      REG_FO2_FIBER_ADDR <= BLOCK_DATA_IN(15 downto 0);
    end if;

    if( WR_FO2_PHASEA = '1' ) then
      REG_FO2_PHASEA <= BLOCK_DATA_IN;
    end if;

    if( WR_FO2_PHASEB = '1' ) then
      REG_FO2_PHASEB <= BLOCK_DATA_IN;
    end if;

    if( WR_FO2_PHASEC = '1' ) then
      REG_FO2_PHASEC <= BLOCK_DATA_IN;
    end if;
   end if;
  end if;
end process;

end Behavioral;
```

9.2.26 DAC.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dac is
    Port (
    -- Block reset for initialization
    BLOCK_RESET_L: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to write to device
    BLOCK_WRITE_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    -- Address in lines for reading address from FPGA
    BLOCK_ADDR_IN: in std_logic_vector( 2 downto 0 );

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    WR_DAC: out std_logic;

    DACLDA: out std_logic;
    DACLDB: out std_logic;
    DACCS: out std_logic;
    DACAB: out std_logic
   );
end dac;

--Address zero - value of HEX displays
--Address 1    - blanking control

architecture Behavioral of dac is
  type FSM_STATE is (idle, active, done);

  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;

  signal COUNTER: integer range 0 to 15;
begin

  --Control state machine
  process( BLOCK_ENABLE, BLOCK_RESET_L, BLOCK_HSCLK )
  begin
--   if( BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0' ) then
--      STATE <= idle;
    if( rising_edge( BLOCK_HSCLK ) ) then
     if(BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0') then
       STATE <= idle;
     else
       STATE <= NEXTSTATE;
     end if;
    end if;
  end process;

  --control device select
  process( BLOCK_HSCLK ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET_L = '0' ) then
       DACLDA <= '0';
       DACLDB <= '0';
       DACAB <= '0';
     else
       DACLDA <= not BLOCK_ADDR_IN(1);
       DACLDB <= BLOCK_ADDR_IN(1);
       DACAB <= BLOCK_ADDR_IN(1);
     end if;
    end if;
  end process;

  --control setup time
  process( STATE, BLOCK_HSCLK ) is
  begin
```

```
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( STATE = active ) then
       COUNTER <= COUNTER + 1;
     else
       COUNTER <= 0;
     end if;
    end if;
  end process;

  --manage state
  process( STATE, BLOCK_WRITE_L, COUNTER )
  begin
    case( STATE ) is
    when idle =>
     BLOCK_DONE <= '1';
     DACCS <= '1';
     NEXTSTATE <= active;
     WR_DAC <= '0';
    when active =>
     BLOCK_DONE <= '0';

     --write to device only if dsp is writing to fpga
     DACCS <= BLOCK_WRITE_L;

     if( COUNTER >= 7 ) then
       NEXTSTATE <= done;
     else
       NEXTSTATE <= active;
     end if;

     WR_DAC <= '1';
    when done =>
     BLOCK_DONE <= '1';
     DACCS <= '1';
     NEXTSTATE <= done;
     WR_DAC <= '1';
    end case;
  end process;

end Behavioral;
```

9.2.27 DBusMux.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dbusmux is
    Port (
    --High Speed clock
    BLOCK_HSCLK: in std_logic;
    --Set to 1 if FPGA is being read by DSP
    BLOCK_READ_L: in std_logic;
    --Set to 1 if FPGA is addressed
    SELECTED: in std_logic;
    --Set to 1 to enable register
    SEL_REG: in std_logic;
    SEL_PESNET: in std_logic;
    --inputs from register block
    REG_IN: in std_logic_vector( 31 downto 0 );

    --inputs from peripheral block
    PBUS_IN: in std_logic_vector( 31 downto 0 );

    --output to DSP data bus
    DATA_OUT: out std_logic_vector( 31 downto 0 );

    PESNET_IN: in std_logic_vector( 31 downto 0 )
   );
end dbusmux;

architecture Behavioral of dbusmux is
signal MUXRESULT: std_logic_vector( 31 downto 0 );
signal DATAEN: std_logic;
signal SEL_REG_KEEP: std_logic;
signal SEL_PESNET_KEEP: std_logic;
begin
  --choose source and create latch
  MUXRESULT <=  REG_IN  when SEL_REG_KEEP = '1' --and
SEL_PESNET = '0'
       else PESNET_IN when SEL_PESNET_KEEP = '1' --and
SEL_PESNET = '0'
       else PBUS_IN ;

  DATAEN <= SELECTED and not BLOCK_READ_L;

  --use latched data for output when read and enable are
active
  DATA_OUT <= MUXRESULT when DATAEN = '1' else
"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

  process( BLOCK_READ_L, BLOCK_HSCLK ) is
  begin
    --if( BLOCK_READ_L  = '1') then
    --  SEL_REG_KEEP <= '0';
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_READ_L = '0' ) then
       SEL_REG_KEEP <= '0';
     else
       SEL_REG_KEEP <= SEL_REG_KEEP or SEL_REG;
     end if;
    end if;
  end process;

  process( BLOCK_READ_L, BLOCK_HSCLK ) is
  begin
--   if( BLOCK_READ_L  = '1') then
--     SEL_PESNET_KEEP <= '0';
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_READ_L = '1' ) then
       SEL_PESNET_KEEP <= '0';
     else
       SEL_PESNET_KEEP <= SEL_PESNET_KEEP or SEL_PESNET;
     end if;
    end if;
  end process;

end Behavioral;
```

9.2.28 DFlash.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dflash is
    Port (
    -- Block reset for initialization
    BLOCK_RESET_L: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to read from device
    BLOCK_READ_L: in std_logic;

    -- Controls whether to write to device
    BLOCK_WRITE_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    --DFLASH signals
    DFLASHOE_L: out std_logic;
    DFLASHWE_L: out std_logic;
    DFLASHCE_L: out std_logic
   );
end dflash;


-- The data is taken care of by the dsp bus.  This
device is mapped onto that bus,
-- eliminating the need for this FPGA to do any work.

architecture Behavioral of dflash is
  type FSM_STATE is (idle, active, done);
  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;

  signal DFLASHCE_LX: std_logic;
begin

process( BLOCK_RESET_L, BLOCK_ENABLE, BLOCK_HSCLK )
begin
--   if( BLOCK_ENABLE = '0' or BLOCK_RESET_L = '0' )
then
--   STATE <= idle;
  if ( rising_edge( BLOCK_HSCLK ) ) then
    if( BLOCK_ENABLE = '0' or BLOCK_RESET_L = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXTSTATE;
    end if;
  end if;
end process;

--manage writes into block
DFLASHWE_L <= BLOCK_WRITE_L or not BLOCK_ENABLE;

--manage reads from block
process( BLOCK_HSCLK, BLOCK_RESET_L, BLOCK_READ_L ) is
variable TRANSACT: std_logic;
begin
  if( rising_Edge( BLOCK_HSCLK )  ) then
    if( BLOCK_READ_L = '1' or BLOCK_RESET_L = '0' ) then
     DFLASHOE_L <= '1';
     TRANSACT := '0';
    else
    --Transact will clear when BLOCK_READ_L rises, and it
will
    --be set when BLOCK_ENABLE goes low
    if( TRANSACT = '1' ) then
     DFLASHOE_L <= '0';
    else
     DFLASHOE_L <= '1';
```

```
    --Begin transaction when block is enabled
    TRANSACT := BLOCK_ENABLE;
   end if;
   end if;
  end if;
end process;

process( BLOCK_HSCLK, BLOCK_RESET_L ) is
variable TRANSACTCE: std_logic;
begin
  if( rising_Edge( BLOCK_HSCLK )) then
   if( BLOCK_RESET_L = '0' ) then
    DFLASHCE_L <= '1';
    TRANSACTCE := '0';
   else
   if( TRANSACTCE = '1' ) then
    TRANSACTCE := (not BLOCK_READ_L ) or (not
DFLASHCE_LX );
    DFLASHCE_L <= '0';
   else
    TRANSACTCE := not DFLASHCE_LX;
    DFLASHCE_L <= DFLASHCE_LX;
   end if;
   end if;
  end if;
end process;


--manage state
process( STATE, BLOCK_READ_L )
begin
  case( STATE ) is
  when idle =>
    BLOCK_DONE <= '1';
    NEXTSTATE <= active;

    DFLASHCE_LX <= '1';
  when active =>
    NEXTSTATE <= done;
    BLOCK_DONE <= '0';

    DFLASHCE_LX <= '0';
  when done =>
    NEXTSTATE <= done;
    BLOCK_DONE <= '1';

    DFLASHCE_LX <= BLOCK_READ_L;
  end case;
end process;

end Behavioral;
```

9.2.29 DIPSW.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dipsw is
    Port (
    -- Block reset for initialization
    BLOCK_RESET: in std_logic;

    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;

    -- Controls whether or not to read from device
    BLOCK_READ_L: in std_logic;

    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;

    -- Latch data before disabling peripheral device
    BLOCK_PDATA_LATCH: out std_logic;

    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;

    DIPTRST: out std_logic
   );
end dipsw;

--Address zero - value of HEX displays
--Address 1    - blanking control
architecture Behavioral of dipsw is
  type FSM_STATE is (idle, active, latched, done );

  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;
  signal COUNTER: integer range 0 to 20;
begin

  --manage clock
  process( BLOCK_ENABLE, BLOCK_RESET, BLOCK_HSCLK )
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET = '0' or BLOCK_ENABLE = '0' ) then
       STATE <= idle;
     else
       STATE <= NEXTSTATE;
     end if;
    end if;  --rising edge
  end process;

  --manage counter
  process( BLOCK_HSCLK )
  begin
    if( rising_Edge( BLOCK_HSCLK ) ) then
     if( STATE = active ) then
       COUNTER <= COUNTER + 1;
     else
       COUNTER <= 0;
     end if;
    end if;
  end process;

  --manage state
  process( STATE, BLOCK_READ_L, COUNTER )
  begin
    case (STATE) is
    when idle =>
     NEXTSTATE <= active;
     DIPTRST <= '1';
     BLOCK_DONE <= '1';
     BLOCK_PDATA_LATCH <= '0';
    when active =>
     if( COUNTER > 10 ) then
       NEXTSTATE <= latched;
     else
       NEXTSTATE <= active;
```

```
    end if;

    DIPTRST <= BLOCK_READ_L;
    BLOCK_DONE <= '0';
    BLOCK_PDATA_LATCH <= '1';  --let data pass through
   when latched =>
    NEXTSTATE <= done;
    DIPTRST <= BLOCK_READ_L;  --only enable if reading
    BLOCK_DONE <= '0';
    BLOCK_PDATA_LATCH <= '0';
   when done =>
    DIPTRST <= '1';
    BLOCK_DONE <= '1';
    NEXTSTATE <= done;
    BLOCK_PDATA_LATCH <= '0';
   end case;
  end process;

end Behavioral;
```

9.2.30 DSP.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dsp is
    Port (
   BLOCK_DONE: out std_logic;
   BLOCK_ENABLE: in std_logic;
   BLOCK_HSCLK: in std_logic;
   BLOCK_RESET: in std_logic;

   DSPSTAT: out std_logic_vector( 7 downto 0 );

   DSPCLK: in std_logic;
   RESET_L: in std_logic;

   DSPCLKEN_L: out std_logic;
   DSPRESET_L: out std_logic;
   --K31

   BMS_L: in stD_logic;
   BRST: in stD_logic;
   CS_L: out std_logic;
   DMAR1_L: out std_logic;
   DMAG1_L: in std_logic;
   DR1: out std_logic;
   DT1: in std_logic;
   DSPTIMEXP_L: in std_logic;
   EBOOT: in std_logic;
   FLAG: out std_logic_Vector( 3 downto 0 );
   HBR_L: out std_logic;
   HBG_L: in std_logic;
   IRQ2_L: inout std_logic;
   IRQ0_L: inout std_logic;
   PA_L: in std_logic;
   PAGE: in stD_logic;
   RCLK1: out std_logic;
   REDY: in std_logic;
   RFS1: out stD_logic;
   RPBA: in std_logic;
   SBTS_L: in std_logic;
   TCK1: in std_logic;
   TFS1: in std_logic

   );
end dsp;

architecture Behavioral of dsp is
type FSM_DSP is ( idle, active, done );
signal STATE, NEXTSTATE: FSM_DSP;

type FSM_DSPRESET is ( idle, active, done );
signal RESET_STATE, RESET_NEXTSTATE: FSM_DSPRESET;

signal DSPSTARTCTR: integer range 0 to 8000;
signal DSPCLKEN_X: std_logic;
begin
DSPCLKEN_L <= not DSPCLKEN_X;


  CS_L    <= '1';
  DMAR1_L <= '1';
  DR1    <= 'Z';
  FLAG    <= "ZZZZ";
  HBR_L  <= '1';
  IRQ2_L  <= '1';
  IRQ0_L  <= '1';
  RCLK1  <= 'Z';
  RFS1    <= 'Z';

process( RESET_L, DSPCLK ) is
begin
  if( rising_edge( DSPCLK ) ) then
    if( RESET_L = '0' ) then
     RESET_STATE <= idle;
    else
     RESET_STATE <= RESET_NEXTSTATE;
    end if;
```

```
    if( RESET_L = '0' ) then
     DSPSTARTCTR <= 0;
    elsif( RESET_STATE = active ) then
     DSPSTARTCTR <= DSPSTARTCTR + 1;
    end if;
  end if;
end process;

process( RESET_STATE, DSPSTARTCTR ) is
begin
  case( RESET_STATE ) is
    when idle =>
     DSPRESET_L <= '0';
     DSPCLKEN_X <= '0';
     RESET_NEXTSTATE <= active;
    when active =>
     DSPRESET_L <= '0';

     if( DSPSTARTCTR < 2048 ) then
       DSPCLKEN_X <= '0';
     elsif( DSPSTARTCTR < 4196 ) then
       DSPCLKEN_X <= '1';
     else
       DSPCLKEN_X <= '0';
     end if;

     --Use 6144 to make the FPGA clock frequency faster
than just using 6000
     if( DSPSTARTCTR < 6144 ) then
       RESET_NEXTSTATE <= active;
     else
       RESET_NEXTSTATE <= done;
     end if;
    when done =>
     DSPRESET_L <= '1';
     DSPCLKEN_X <= '0';
     RESET_NEXTSTATE <= done;
  end case;
end process;
```

```
  process( BLOCK_RESET, BLOCK_ENABLE, BLOCK_HSCLK ) is
  begin
    --if( BLOCK_RESET = '0' or BLOCK_ENABLE = '0' ) then
    --  STATE <= idle;
    if( rising_Edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET = '0' or BLOCK_ENABLE = '0' ) then
       STATE <= idle;
     else
       STATE <= NEXTSTATE;
     end if;
     end if;
  end process;

  process( STATE ) is
  begin
    case (STATE) is
    when idle =>
     BLOCK_DONE <= '1';
     NEXTSTATE <= active;
    when active =>
     BLOCK_DONE <= '0';
     NEXTSTATE <= done;
    when done =>
     BLOCK_DONE <= '1';
     NEXTSTATE <= done;
    end case;
  end process;

  DSPSTAT(7) <= TFS1 and DT1 and TCK1 ;
  DSPSTAT(6) <=  BRST and SBTS_L and PA_L and DMAG1_L
and REDY and HBG_L;
  DSPSTAT(5) <=  PAGE and DSPTIMEXP_L;
  DSPSTAT(4) <=  EBOOT and RPBA;
  DSPSTAT(3 downto 0 ) <= "0000";

end Behavioral;
```

9.2.31 Hex.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hexdisp is
    Port (
    -- Block reset for initialization
    BLOCK_RESET: in std_logic;
    -- DSP is selecting the peripherals
    BLOCK_ENABLE: in std_logic;
    -- Controls whether or not to read from device
    BLOCK_WRITE_L: in std_logic;
    -- Synchronization line indicating that the block is
working when it is low
    BLOCK_DONE: out std_logic;
    --high speed clock (160 MHz)
    BLOCK_HSCLK: in std_logic;
    WR_HEX: out std_logic;
    HEXLATCH: out std_logic
   );
end hexdisp;

architecture Behavioral of hexdisp is
  type FSM_STATE is ( idle, active, latched, done);

  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;

  signal COUNTER: integer range 0 to 15;
begin

  --update counter
  process( BLOCK_ENABLE, BLOCK_RESET, BLOCK_HSCLK )
  begin
    --if( BLOCK_RESET = '0' or BLOCK_ENABLE = '0') then
    --  STATE <= idle;
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( BLOCK_RESET = '0' or BLOCK_ENABLE = '0' ) then
       STATE <= idle;
     else
       STATE <= NEXTSTATE;
     end if;
    end if;  --rising edge
  end process;

  --manage counter
  process( BLOCK_HSCLK, STATE ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( STATE = active ) then
       COUNTER <= COUNTER + 1;
     else
       COUNTER <= 0;
     end if;
    end if;
  end process;

  --manage state
  process( STATE , BLOCK_WRITE_L, COUNTER ) is
  begin
  case( STATE ) is
    when idle =>
     HEXLATCH <= '1';
     NEXTSTATE <= active;
     BLOCK_DONE <= '1';
     WR_HEX <= '0';
    when active =>
     BLOCK_DONE <= '0';
     HEXLATCH <= BLOCK_WRITE_L;
     WR_HEX <= '1';

     if( COUNTER >= 7 ) then
       NEXTSTATE <= latched;
     else
       NEXTSTATE <= active;
     end if;
    when latched =>
     BLOCK_DONE <= '0';
```

```
      HEXLATCH <= '1';
      NEXTSTATE <= done;
      WR_HEX <= '1';
     when done =>
      BLOCK_DONE <= '1';
      HEXLATCH <= '1';
      NEXTSTATE <= done;
      WR_HEX <= '0';
  end case;
  end process;

end Behavioral;
```

9.2.32 PBusCtl.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--This block controls the peripheral data bus
entity PBUSCTL is
    Port (
   BLOCK_HSCLK: in std_logic;

    --device enable from selector
   EN_DIPSW: in std_logic;
   EN_HEX: in std_logic;
   EN_PMC: in std_logic;
   EN_DAC: in std_logic;

   --Latch input data from peripheral bus
   LATCH_PMC: in std_logic;
   LATCH_DIPSW: in std_logic;

   PREAD_PMC: in std_logic;

   PWRITE_PMC: in std_logic;

   --address line controls
   ADDR_PMC: in std_logic_vector( 18 downto 0 );

   --output enables for data bus
   WR_PMC: in std_logic;
   WR_HEX: in std_logic;
   WR_DAC: in std_logic;


   --Signals going out to bus
   PREAD: out std_logic;
   PWRITE: out std_logic;
   PADDR: out std_logic_vector( 18 downto 0 );
   PDATA: inout std_logic_vector( 31 downto 0 );

   --Input from DSP
   DSP_DATA_IN: in std_logic_vector( 31 downto 0 );

   --Output to FPGA blocks
   PDATA_IN: out std_logic_vector( 31 downto 0 )
  );
end PBUSCTL;

architecture Behavioral of PBUSCTL is
signal LATCH_EN: std_logic;
signal OUTPUT_EN: std_logic;
begin
  PREAD <= PREAD_PMC when EN_PMC = '1' else 'Z';
  PWRITE <= PWRITE_PMC when EN_PMC = '1' else 'Z';

  --Control data flow on bus output
  OUTPUT_EN <= (WR_PMC and EN_PMC ) or (WR_DAC and
EN_DAC) or (WR_HEX and EN_HEX);

  PDATA <= DSP_DATA_IN when OUTPUT_EN = '1' else
"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
  --Latch the data from the peripheral bus
  LATCH_EN <= (LATCH_PMC and EN_PMC ) or (LATCH_DIPSW
and EN_DIPSW );

  process( BLOCK_HSCLK, LATCH_EN ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
     if( LATCH_EN = '1' ) then
        PDATA_IN <= PDATA;
     end if;
    end if;
  end process;

  --control peripheral address bus
  PADDR <= ADDR_PMC when EN_PMC = '1' else
"ZZZZZZZZZZZZZZZZZZZ";
end Behavioral;
```

9.2.33 PlaceHolder.VHD

```
--simulates a block that hasn't been made yet.  It just
pulses done to make it look like it did something.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity placeholder is
    Port (
    RESET_L: in std_logic;
    BLOCK_ENABLE: in std_logic;
    BLOCK_DONE: out std_logic;
    HSCLK: in std_logic
   );
end placeholder;

architecture Behavioral of placeholder is
  type FSM_STATE is (idle, active, done );

  signal STATE: FSM_STATE;
  signal NEXTSTATE: FSM_STATE;
begin

P0: process( RESET_L, BLOCK_ENABLE, HSCLK )
begin
--  if( RESET_L = '0' or BLOCK_ENABLE = '0') then
--   STATE <= idle;
  if( rising_edge(HSCLK) ) then
    if(RESET_L = '0' or BLOCK_ENABLE = '0' ) then
     STATE <= idle;
    else
     STATE <= NEXTSTATE;
    end if;
  end if;
end process;

process( STATE )
begin
  case ( STATE ) is
    when idle =>
     NEXTSTATE <= active;
     BLOCK_DONE <= '1';
      when active =>
     NEXTSTATE <= done;
     BLOCK_DONE <= '0';
    when done =>
     NEXTSTATE <= done;
     BLOCK_DONE <= '1';
  end case;
end process;
end Behavioral;
```

9.2.34 PMC.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity PMC is
  port (
        BLOCK_HSCLK      : in std_logic; --PCI Clock
(40MHz)
        BLOCK_ENABLE       : in std_logic; --PMC Block
Enable
        BLOCK_RESET_L    : in std_logic; --PMC Block
Reset
        BLOCK_DONE    : out std_logic; --PMC Block Busy
        BLOCK_READ_L     : in std_logic; --DSP read low
strobe
        BLOCK_WRITE_L     : in std_logic; --DSP write
low strobe

        blast_l      : out std_logic; --DP Burst Last
        data_en      : out std_logic; --Enables data bus
from FPGA to DP
        dp_rd_l      : out std_logic; --DP Read
        dp_wr_l      : out std_logic; --DP Write
     reg_en    : out std_logic;  --register latch enable

        PMCCLK       : in std_logic;
     rdy_out_l    : in std_logic; --Ready Out signal
from DP RAM
        ale          : out std_logic; --DP Address Latch
Enable
        rdy_in       : out std_logic; --DP Ready In
        select_l     : out std_logic; --DP Select
        strobe_l     : out std_logic; --DP Strobe
        test         : out std_logic); --DP Test
end pmc;

-- Sequential architecture of Dual Port (DP) Interface
-- Interfaces DSP to DP RAM
```

```
architecture RTL of PMC is
  -- Components
     -- None
  -- Internal signals
  type dp_state is ( init, rd_addr, rd_data, latch_data,
wr_addr, wr_data, wr_done );
  signal cur_state    : dp_state; --The current state of
the DP
  signal new_state    : dp_state; --The next state of
the DP
  signal reg_enx       : std_logic; --Data Latch Enable
  signal WATCHDOG      : integer range 0 to 15;
  signal WATCHDOG_FIRE: std_logic;
  -- Component instantiations
     -- None
  -- Netlist/Concurrent Statements
begin

reg_en <= not reg_enx;

blast_l <= '0';
rdy_in <= '1';
ale <= '1'; --keep active
test <= '0';
dp_wr_l <= BLOCK_WRITE_L;
dp_rd_l <= '1'; --not used in default config



process( cur_state, BLOCK_HSCLK ) is
begin
-- if( cur_state /= rd_data and cur_state /= wr_data )
then
--   WATCHDOG <= 0;
--   WATCHDOG_FIRE <= '0';
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( cur_state /= rd_data and cur_state /= wr_data )
then
     WATCHDOG <= 0;
     WATCHDOG_FIRE <= '0';
    else
```

```
    WATCHDOG <= WATCHDOG + 1;

    if( WATCHDOG >= 8 ) then
      WATCHDOG_FIRE <= '1';
    end if;
   end if;
  end if;
end process;

    --Dual Port Write and Read State Machine [FSM]
process(cur_state, BLOCK_ENABLE, BLOCK_READ_L,
BLOCK_WRITE_L, rdy_out_l, watchdog_fire)
begin
  case cur_state is
  when init =>
    select_l <= '1';
    strobe_l <= '1';

    data_en <= '0';
    BLOCK_DONE <= '1';
    reg_enx <= '0'; --Allow Data through
    data_en <= '0';

    if BLOCK_READ_L = '0' then
     new_state <= rd_addr;
    elsif BLOCK_WRITE_L = '0' then
     new_state <= wr_addr;
    else
     new_state <= init;
    end if;

  when rd_addr =>
    strobe_l <= '0';
    select_l <= '0';
    BLOCK_DONE <= '0';
    reg_enx <= '0'; --Latch data from DP RAM
    data_en <= '0';
    new_state <= rd_data;

  when rd_data =>
    strobe_l <= '1';
    select_l <= '0';
    BLOCK_DONE <= '0';
    reg_enx <= '0'; --Latch data from DP RAM
    data_en <= '0';

    if rdy_out_l = '0' or WATCHDOG_FIRE = '1' then
     new_state <= latch_data;
    else
     new_state <= rd_data; --Waiting for rdy_out_l from
DP
    end if;

  when latch_data =>
    select_l <= '0';
    strobe_l <= '1';
    reg_enx <= '1'; --Latch data from DP RAM
    data_en <= '0';
    new_state <= latch_data;
           BLOCK_DONE <= '1';
    reg_enx <= '0';

  when wr_addr =>
    strobe_l <= '0';
    select_l <= '0';
    BLOCK_DONE <= '0';
    data_en <= '1';
    reg_enx <= '0';
    new_state <= wr_data;

  when wr_data =>
    strobe_l <= '1';
    select_l <= '0';
    BLOCK_DONE <= '0';
    data_en <= '1';
    reg_enx <= '0';

    if( rdy_out_l = '0' or WATCHDOG_FIRE = '1') then
     new_state <= wr_done;
    else
     new_state <= wr_data;
    end if;

  when wr_done =>
```

```
    strobe_l <= '1';
    select_l <= '1';
    BLOCK_DONE <= '1';
    data_en <= '1';
    new_state <= wr_done;
    reg_enx <= '0';

  end case;
end process;

    --Update State
process(BLOCK_HSCLK, BLOCK_ENABLE, BLOCK_RESET_L)
variable LOCKED: std_logic;
begin
  if rising_edge( BLOCK_HSCLK) then
    if BLOCK_RESET_L = '0' or BLOCK_ENABLE = '0' then
     cur_state <= init;
     LOCKED := '0';

    elsif( PMCCLK = '0' and LOCKED /= '1' ) then
     cur_state <= new_state;
     LOCKED := '1';
    elsif( PMCCLK = '1' ) then
     LOCKED := '0';
    end if;
  end if;
end process;

end rtl;
```

9.2.35 RA_PWM.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the
declarations that are
--  provided for instantiating Xilinx primitive
components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RA_PWM is
port(
  BLOCK_HSCLK: in std_logic;
  BLOCK_RESET_L: in std_logic;

  DUTY_COUNT: in std_logic_vector( 15 downto 0 );
  PWM_SAT: in std_logic_vector( 15 downto 0 );
  SYNC: in std_logic;

  PWM_OUT: out std_logic
);
end RA_PWM;

architecture Behavioral of RA_PWM is
signal CTR: std_logic_vector( 15 downto 0 );
signal SAT: std_logic;
signal SYNC_SUPPRESS: std_logic;

begin

process( BLOCK_HSCLK ) is
begin
  if( rising_edgE( BLOCK_HSCLK ) ) then
    if( CTR < PWM_SAT ) then
     SAT <= '0';
    else
     SAT <= '1';
    end if;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edge( BLOCK_HSCLK ) ) then
    if( SYNC = '1' and SYNC_SUPPRESS = '0') then
     CTR <= X"0000";
    elsif( SAT = '0' ) then
     CTR <= CTR + 1;
    end if;

    --generate one cycle lag
    SYNC_SUPPRESS <= SYNC;
  end if;
end process;

process( BLOCK_HSCLK ) is
begin
  if( rising_edgE( BLOCK_HSCLK ) ) then
    if( CTR < DUTY_COUNT ) then
     PWM_OUT <= '1';
    else
     PWM_OUT <= '0';
    end if;
  end if;
end process;

end Behavioral;
```

9.2.36 Selector.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SELECTOR is
    Port (
     --High speed clock
    BLOCK_HSCLK: in std_logic;
    BLOCK_RESET_L: in std_logic;
    BLOCK_READ_L: in std_logic;
    BLOCK_WRITE_L: in std_logic;

    --FPGA ID
    ID: in std_logic_vector( 2 downto 0 );

    --DSP Address bus
    ADDR: in std_logic_vector( 31 downto 25 );
     ACK: inout std_logic;  --ACK is pull-up
    BMS_L: in std_logic;

    --Device select lines.  Only one of these should be
on.  These are high when the device is selected
    EN_DAC: out std_logic;
    EN_DIPSW: out stD_logic;
    EN_DSP: out std_logic;
    EN_FPGA: out stD_logic;
    EN_HEX: out std_logic;
    EN_PERIPH: out std_logic;
    EN_PEEPROM: out std_logic;
    EN_PESNET: out std_logic;
    EN_PMC: out std_logic;
    EN_ULCTL: out stD_logic;
    EN_DFLASH: out std_logic;
--NEW MODULE: Insert EN_MODULE here

    --These go low when the device is working
    DONE_DAC: in std_logic;
    DONE_DIPSW: in std_logic;
    DONE_DSP: in std_logic;
    DONE_FPGA: in std_logic;
    DONE_HEX: in std_logic;
    DONE_PERIPH: in stD_logic;
    DONE_PEEPROM: in stD_logic;
    DONE_PESNET: in stD_logic;
    DONE_PMC: in stD_logic;
    DONE_ULCTL: in std_logic;
    DONE_DFLASH: in std_logic;
--NEW MODULE: Insert DONE_MODULE here

    --These are used when the FPGA wants to talk to the
DSP first
    --Examples of this usage would be a request from PMC
or Fiber
    --When in host bus access mode, the selector ignores
its inputs until
    --the host bus request line goes low.
    --In order to get in this mode, the DSP block will
set HOST_BUS_ACCESS_REQUEST
    --to a 1, and then wait for HOST_BUS_ACCESS_GRANT to
go to a 1.  When it goes to a 1,
    --the rest of the FPGA can ensure that the selector
does not have access to the DSP bus. during this time,
    --the DSP control block can use the host bus grant
lines to access DSP internal memory and perform
    --transactions with the DSP.
    HOST_BUS_ACCESS_REQUEST: in std_logic;
    HOST_BUS_ACCESS_GRANT: out std_logic;

    SELECTED: out std_logic --When 1, then the FPGA is
addressed
   );
end SELECTOR;

architecture Behavioral of SELECTOR is

  type FSM_SEL is ( idle, waitbusy, waitdone, done);
```

```
  signal SEL_STATE: FSM_SEL;
  signal SEL_NEXTSTATE: FSM_SEL;

  type FSM_DEV is ( none, dac, dipsw, dsp, fpga, hex,
peeprom, periph, pesnet, pmc, ulctl, dflash );
  signal DEV_EN: FSM_DEV;

  signal BUSY_L: std_logic;

  signal SELECTEDX: std_logic;

  constant ADDR_DAC:     std_logic_vector( 3 downto 0 )
:=  "0001";
  constant ADDR_DIPSW:   std_logic_vector( 3 downto 0 )
:=  "0010";
  constant ADDR_DSP:     std_logic_vector( 3 downto 0 )
:=  "1011";
  constant ADDR_FPGA:    std_logic_vector( 3 downto 0 )
:=  "0100";
  constant ADDR_HEX:     std_logic_vector( 3 downto 0 )
:=  "0101";
  constant ADDR_PERIPH:  std_logic_vector( 3 downto 0 )
:=  "0110";
  constant ADDR_PEEPROM: std_logic_vector( 3 downto 0 )
:=  "0111";
  constant ADDR_PESNET:  std_logic_vector( 3 downto 0 )
:=  "1000";
  constant ADDR_PMC:     std_logic_vector( 3 downto 0 )
:=  "1001";
  constant ADDR_ULCTL:   std_logic_vector( 3 downto 0 )
:=  "1010";
  constant ADDR_DFLASH:  std_logic_vector( 3 downto 0 )
:=  "1100";
  constant ADDR_DEBUG:   std_logic_vector( 3 downto 0 )
:=  "1111";

begin
  --indicate that the device is selected
  process( BLOCK_HSCLK ) is
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
      if( (ID(2 downto 0 ) = ADDR( 31 downto 29 )) and (
BLOCK_WRITE_L = '0' or BLOCK_READ_L = '0' )) then
        SELECTEDX <= '1' ;
      else
        SELECTEDX <= '0';
      end if;
    end if;
  end process;

  SELECTED <= SELECTEDX;

  BUSY_L <= DONE_DAC and DONE_DIPSW  and DONE_FPGA and
DONE_HEX and DONE_PERIPH and DONE_PEEPROM and
DONE_PESNET and DONE_PMC and DONE_ULCTL and DONE_DFLASH
and DONE_DSP;

  --for now, disable this
  HOST_BUS_ACCESS_GRANT <= '1';

  ACK <= '0' when SEL_STATE = waitbusy or SEL_STATE =
waitdone else '1';

  --Control state machine
  process( BLOCK_HSCLK, BLOCK_RESET_L, SELECTEDX )
  begin
    if( rising_edge( BLOCK_HSCLK ) ) then
      if( BLOCK_RESET_L = '0' or SELECTEDX = '0' ) then
        SEL_STATE <= idle;
      else
        SEL_STATE <= SEL_NEXTSTATE;
      end if;
    end if;
  end process;

  EN_DAC    <= '1' when DEV_EN = dac            else '0';
  EN_DIPSW  <= '1' when DEV_EN = dipsw   else '0';
  EN_DSP    <= '1' when DEV_EN = dsp            else '0';
  EN_FPGA   <= '1' when DEV_EN = fpga           else '0';
  EN_HEX    <= '1' when DEV_EN = hex            else '0';
  EN_PEEPROM  <= '1' when DEV_EN = peeprom      else '0';
  EN_PERIPH   <= '1' when DEV_EN = periph       else '0';
  EN_PESNET   <= '1' when DEV_EN = pesnet       else '0';
```

```
  EN_PMC    <= '1' when DEV_EN = pmc            else '0';
  EN_ULCTL  <= '1' when DEV_EN = ulctl    else '0';
  EN_DFLASH   <= '1' when DEV_EN = dflash       else '0';


  process( SELECTEDX, BLOCK_HSCLK )
  begin
--   if( SELECTEDX = '0' ) then
--     DEV_EN <= none;
    if( rising_edge( BLOCK_HSCLK ) ) then
    if(SELECTEDX = '0') then
     DEV_EN <= none;
    else
     if( SELECTEDX = '1' ) then
       case( SEL_STATE ) is
         when waitbusy |  waitdone | idle =>
           if( BMS_L = '0' ) then
            DEV_EN <= dflash;
           else
            case ADDR( 28 downto 25 ) is
              when ADDR_DAC =>           DEV_EN <= dac;
              when ADDR_DIPSW =>  DEV_EN <= dipsw;
              when ADDR_FPGA =>   DEV_EN <= fpga;
              when ADDR_HEX =>    DEV_EN <= hex;
              when ADDR_PERIPH =>  DEV_EN <= periph;
              when ADDR_PEEPROM => DEV_EN <= peeprom;
              when ADDR_PESNET =>  DEV_EN <= pesnet;
              when ADDR_PMC =>    DEV_EN <= pmc;
              when ADDR_ULCTL =>  DEV_EN <= ulctl;
              when ADDR_DSP =>    DEV_EN <= dsp;
              when ADDR_DFLASH => DEV_EN <= dflash;
              when ADDR_DEBUG => DEV_EN <= none; --
DEBUGGING (reserved for now)
              when others => null;
            end case;
           end if;
         when others =>  --other selstate
           DEV_EN <= none;
       end case;
     end if;
     end if;
    end if;
```

```
  end process;

  --Decode and select device
  process( SEL_STATE, BUSY_L, BLOCK_READ_L,
BLOCK_WRITE_L, SELECTEDX, BMS_L, ADDR, BLOCK_RESET_L,
DEV_EN )
  begin
  case (SEL_STATE) is
  when idle =>
    SEL_NEXTSTATE <= waitbusy;

  when waitbusy =>
    if( DEV_EN = none ) then
     SEL_NEXTSTATE <= done;
    elsif( BUSY_L = '0' ) then
     SEL_NEXTSTATE <= waitdone;
    else
     SEL_NEXTSTATE <= waitbusy;
    end if;

  when waitdone =>
    if( BUSY_L = '1' ) then
     SEL_NEXTSTATE <= done;
    else
     SEL_NEXTSTATE <= waitdone;
    end if;

  when done =>
    SEL_NEXTSTATE <= done;
  end case;
  end process;
end Behavioral;
```

10 Vita

Gerald Francis was born in Glen Cove, New York on May 3rd, 1978. He came to Virginia Polytechnic Institute and State University (Virginia Tech) in the Fall semester of 1996. He graduated with a Bachelor of Science degree in Computer Engineering in May 2001. From 1997 to 1998, he completed a three semester co-op for Wonderware Corp. (now a member of Invensys Plc.) during which he focused on quality assurance / automated testing and software development for a software based programmable logic controller (PLC) running on Windows NT and Windows CE. In 1999, the author worked for Exegetics Inc. as a quality assurance technician and system administrator. In 2000, Gerald joined the Center for Power Electronics Systems at Virginia Tech as an undergraduate research assistant, where he assisted researchers on digital control for a soft-switched inverter.

In 2001, Gerald became a Master of Science student in Electrical Engineering with a graduate research assistantship at CPES. He has worked there since, sponsored by the Office of Naval Research and the National Science Foundation. Gerald defended his master's thesis on March 3, 2004. In the summer of 2004, Gerald took an internship at General Dynamics Advanced Information systems as a result of this research. During his internship, he implemented the majority of PESNet version 2.2.

Gerald is currently a Ph.D. candidate at Virginia Tech in Electrical Engineering under the direction of Dr. Dushan Boroyevich. His current research focuses on three phase system stability, and is currently sponsored by The Boeing Company and the National Science Foundation.