1997

Development and analysis of a verstile, reusable, high speed, DMA controller for custom embedded applications using the PCI bus

Michael Eastman

Rochester Institute of Technology Computer Science Department

Development and Analysis of a Versatile, Reusable, High Speed, DMA Controller for Custom Embedded Applications using the PCI Bus

by
Michael G. Eastman

05/05/97

A Thesis
submitted to
The Faculty of the Computer Science Department
at the
Rochester Institute of Technology
by
Michael G. Eastman

Approved by:

Doctor Peter G. Anderson

Doctor Jim Heliotis

Professor George H. Zion

May 5, 1997

Title of Thesis: Development and Analysis of a Versatile, Reusable, High Speed DMA Controller for Custom Embedded Applications

1. ABSTRACT

This thesis investigates the plausibility of designing and developing a versatile, reusable, high speed interface for custom computing applications, based on the Peripheral Component Interface (PCI) Bus. A PCI I/O board was developed, utilizing mainly Complex Programmable Logic Devices (CPLD's), which included a custom Direct Memory Access (DMA) Controller to take advantage of the unique feature set of the PCI bus. The arbitration mechanisms and performance characteristics of the PCI bus are taken advantage of in order to achieve a maximum burst throughput rate of 66 Megabytes per second. Performance characteristics of the I/O board are analyzed for two separate PCI host systems. In the faster of the two systems, a 166MHz Pentium PC, a maximum aggregate throughput rate of 54 Megabytes per second for PCI burst writes was achieved. In all cases throughput increased as a function of transfer size. Due to buffering implementations in the host systems write performance was always superior to read performance.

In addition to exceptional throughput capability, this implementation provides a design engineer with a versatile interface which can be mated to a number of high performance applications. The PCI I/O board's external interface is implemented with a CPLD which can be quickly and easily modified to meet the needs of practically any custom interface without decreasing PCI bus performance. Using the on-board latency timer and programmable FIFO's the board can be fine tuned to meet a variety of application requirements.

The two main design goals were to provide unlimited bursting capability and to transfer 32-bits of data on every clock. The first was achieved through the implementation of a 32-bit burst Transfer Count register. The second goal had to be reduced by 50% due to a timing margin violation discovered during board debug.

2. Introduction and Background

2.1. Platform Selection

Many of today's computer based imaging applications, such as printing, scanning, and video rendering, require large amounts of data to be moved very quickly. The main requirement is to move data between a main memory resource (typically Dynamic Random Access Memory (DRAM)) and a specific, often custom, Input/Output (I/O) Interface. This thesis provides a Personal Computer (PC) based mechanism for high speed data transfer between system DRAM and a first-in-first-out (FIFO) Memory system. To perform this task a Peripheral Component Interconnect (PCI) add-in card was developed which may be easily interfaced to a variety of custom I/O subsystems using programmable logic. In fact a logic device has been provided which may be modified to meet a designers specific needs. It should be noted that although this design is targeted for a PC add-in card the PCI bus has spread to a variety of other systems with the same electrical characteristics but different mechanical requirements. There are numerous examples such as PCI Mezzanine Card (PMC) which is used for industrial bus (VME, Multibus) expansion, PowerPC platforms from Apple, Motorola, or IBM, etc.. This design could be easily reused for these other system architectures without modifying any of the logic design. A modification of the mechanical form-factor would enable this design to function gracefully in these environments.

An embedded systems designer has a wide range of "platform" or Bus Choices today. Multibus II, Multibus, and VME are a few of the more prevalent "industrial" busses currently available. The problem with systems based on these busses is that they tend to be expensive and often have long development cycles. It becomes necessary to purchase multiple boards for various system requirements, or to design your entire system on a single board or set of boards. The PC, however, provides a wide range of commonly used features such as parallel ports, serial ports, and high quality video graphics. Recently Personal Computers have been migrating into a larger number of industrial or embedded applications. This is due to the fact that PC's are becoming more and more ubiquitous causing their price/performance ratio to become smaller and smaller which in turn makes them much more attractive to designers of low volume applications. This trend has, over the last few years, enabled Original Equipment Manufacturers (OEMs) to leverage the PC volume manufacturing for a growing number of embedded systems. Their utilization covers the full spectrum, from using PC-type Chip-sets and microprocessors and designing them onto a custom board within a system, to actually embedding an entire PC (with or without chassis) into a larger system. In all cases the goal is to take advantage of the hardware price/performance and/or the vast amount of software that is currently available for PC-type machines. One of the compelling reasons for utilizing a PC in an embedded environment is the availability and ease of use of a Graphical User Interface (GUI). Historically when

designing an embedded system one of the most difficult tasks has been implementing an elegant operator interface. Development of graphical interfaces was typically custom leading to interface models which were difficult to develop, and often cumbersome to use. Additionally there was no standard look or feel between applications. With a PC Platform the Video interface is free, along with widely available development environments for graphical interfaces.

In the past, the main drawback of the PC for many embedded applications was limited performance. Today's Industry Standard Architecture (ISA) Bus has remained essentially the same since IBM came out with the PC-AT and has a maximum bandwidth of just over five Megabytes Per Second[1] Additionally, having multiple masters of an ISA Bus is cumbersome and inefficient. In order to perform DMA type operations the Central Processing Unit (CPU) was put into a hold state and no concurrent operations could take place between the CPU and alternate masters. The enhanced version of this bus (developed by a consortium led by COMPAQ) which increased the data bus width from 16 to 32 bits and was named EISA, for Enhanced ISA, has a theoretical bandwidth of 33 Megabytes Per Second, and provides a somewhat more robust method of handling multiple masters[2] The advent of the PCI Bus, which is 32 bits wide and has a theoretical maximum bandwidth of 132 Megabytes Per Second, has enabled embedded system designers to utilize the Personal Computer in a much larger number of applications which require higher bandwidth. High speed burst transactions and multiple masters are gracefully provided for in the PCI Specification. The PCI Bus architecture was developed in such a way that it is decoupled from the CPU thus enabling concurrent operation between the CPU and alternate PCI masters.

One of the most common areas requiring high data throughput for embedded systems is imaging. Video applications must be clear and crisp and look pleasing to the eye. Documents and printing plates must be rendered rapidly, and often one document must be scanned in while another one is being printed out, and perhaps another one is being manipulated by a variety of image processing algorithms in preparation for print-out. Often there may be multiple events occurring which each require a throughput of greater than 20 Megabytes Per Second. This is clearly beyond the capability of either ISA or EISA. In the past the Personal Computer was not an option for this type of application. Today the PCI Bus provides a robust interface which enables multiple masters to utilize the high bandwidth bus.

2.2. Design Goals

The overall objective of this investigation was to design and implement a PCI Add-in-Board based upon a custom Direct Memory Access (DMA) Controller. This technique was chosen because a DMA Controller requires less programming overhead than a CPU solution, is less complex, and more efficient for data

movement operations. The most unique aspect of the DMA Controller designed for this thesis is the ability to perform very large numbers of data transactions per burst. Most typical DMA Controllers provide burst counts of four or less. The performance goal was to operate at a burst data transfer rate approaching the theoretical maximum throughput bandwidth of the PCI Bus. The aggregate rate would be somewhat slower depending upon bandwidth utilization by other potential PCI Bus initiators (e.g. Host CPU, other PCI add-in cards, etc.). Actual system throughput is ultimately determined by the performance of the PCI implementation and DRAM Subsystem within the host PC System as well as the efficiency of this design.

In addition to throughput performance, a goal of this development was to provide a solution which required very little CPU programming overhead. If the software necessary to control the DMAC was too complex and cumbersome the performance achieved may not be worth the overhead involved. To accomplish this goal, a small effective register set was implemented. Additionally both Normal and Chained DMA transfers are supported to further reduce software complexity.

A secondary goal of this thesis was to enable other design engineers to utilize the PCI portion of this design and customize the I/O interface to their own specific requirements. A functional interface was designed for evaluation purposes, but ultimately the PCI I/O board must be malleable enough to be incorporated into an actual application. More than twenty-two signals, including 8-data bits, are available within a single CPLD device to provide a design engineer with resources to implement a robust custom parallel interface.

Another unique aspect of this implementation is that it is suitable for applications which are considered low volume and high performance in the personal computer industry. In many instances an off-the-shelf solution is not suitable, and developing an ASIC is cost prohibitive. The solution proposed here is more expensive than an off-the-shelf solution, but much lower cost than an ASIC implementation for small volumes. For an application which would utilize hundreds of thousands of units, an ASIC may indeed be a less expensive overall solution. This implementation could be ported to an ASIC design if so desired. This thesis focuses on the DMA Controller and PCI Bus Interface Implementation including the data path from the PCI Bus to the on-board FIFO Subsystem. Details of some of the surrounding logic such as the PCI Configuration Region, which is a general PCI requirement and not inherently unique, are considered beyond the scope of this thesis.

2.3. Technical Overview

This section provides the details of the development portion of this thesis. It is broken down into two subsections Hardware Development, and Validation Software. The goals and reasons for the thesis are described.

2.3.1. Custom Hardware

Complex Programmable Logic Devices (CPLD's) manufactured by Altera Corporation were selected as the logical interface to the PCI Bus and as the basis for a custom DMA Controller developed explicitly for this application. The PCI add-in card does not contain a microprocessor, but is programmed by the Personal Computer's host processor. Once programmed, the DMA Controller acts as a PCI Initiator or Bus Master. It has the ability to initiate PCI Bus Cycles and perform high speed data transfers between main system memory and the on-board FIFO memory buffer data store. Once on the PCI Interface Board Data is moved between the FIFO and a custom bi-directional I/O interface designed specifically to demonstrate the functionality of this approach. The FIFO memory may be read from and written to at the same time. For example, when a DMA transfer is taking place such that data is being written from system memory into the on-board FIFO's, data may also be moved from the on-board FIFO's to an external device such as a laser marking engine. In a "real-life" application the I/O interface would be designed to match whatever device the PCI add-in card was delivering data to or receiving data from. The intent here was to utilize the bandwidth available on the PCI bus in as efficient a manner as possible assuming that the device developed here was the highest priority device in the system. This requirement precluded the use of any currently available off-the-shelf DMA Controllers. This requirement meant that a device specifically designed to operate with the PCI bus was required. Some of the unique performance features of the DMA Controller developed for this thesis are:

- Seamless Logical Interface to the PCI Bus
- Virtually Unlimited Bursting Capability (4 Gig Burst Count)
- Fly-by mode optimized for DRAM-FIFO Interface
- 4 Gigabyte Addressing Capability

The items designed and/or developed for the Hardware Development portion of this thesis are as follows:

1) *Hierarchical Schematic Entry for board design*
2) *Logic Design & Syntehesis of PCI DMA Controller Functionality*
3) *Logic Design & Synthesis of PCI Burst Bus Master & Slave Interfaces*

4) *Logic Design & Synthesis of FIFO Input/Output Interface*

5) **Layout, Routing, and Manufacture of Printed Circuit Board*

6) *Debug and Analysis of Printed Circuit Board*

*This is a Purchased Service handled by a manufacturing facility.

2.3.2. Validation Software

The Validation software consists of a series of tests to show how the PCI I/O board performs from a data movement standpoint. These tests demonstrate functionality in addition to enabling performance measurement. There is a test for each one of the four possible modes of operation for the board as follows:

- Normal Mode Input
- Normal Mode Output
- Chaining Mode Input
- Chaining Mode Output

The direction of the each of the Modes refers to the direction of data movement with respect to an external device. In input mode data is transferred from the on-board FIFO memory subsystem to the host PC's DRAM subsystem (Memory Write Operation with respect to the PCI Bus). In output mode data is transferred from DRAM to the on-board FIFO memories (Memory Read Operation with respect to the PCI Bus). The objective of the software is to provide a mechanism to evaluate functionality and overall board performance. The software is written in such a manner that the PCI I/O board will attempt to operate as fast as possible for user specified data sizes. Since the data transfer size capability of the add-in card is 4Gigabytes, the actual data transfer size is bounded only by the amount of memory in the PC. Actual throughput is measured with both a logic analyzer and with an internal software timer. This correlation demonstrates the accuracy of the performance analysis.

The software utilizes the on-board PCI Configuration Register information to determine the base address of the PCI I/O board. The on-board DMA Controller is then programmed to perform a specified number of 32-bit wide transfers between main memory and the PCI add-in board. The direction may be selected as input (as would be the case for a scanner type operation) or as output (as would be the case for a marking type operation). The DMA transfer process is initiated by programming the DMA Control register in either chaining or normal mode as described in the preceding sections. Upon completion of the DMA process an interrupt provides notification that the transfer has finished.

The goal of the evaluation process is not to investigate or analyze operating system latency issues, but to simply determine the performance of the hardware subsystem within a given PC environment. Since the PCI I/O board has no microprocessor, and therefore no software to execute, it is beyond the scope of this investigation to attempt to analyze operating system dependencies of the host system being tested. Additionally the portion of the design that is performance critical is the board's ability to move data between system memory and the local on-board FIFO's. The interface between the on-board FIFO's and the external world (scanner, marking engine, etc.) is typically custom and/or specific to the particular interface device. The external interface for this demonstration vehicle is implemented as a simple 8-bit wide differential handshaken bus. For this design to be used in a real application it would be the designer's responsibility to customize that portion of the design to interface requirements of the peripheral being accessed. This interface could be any number of bits wide depending upon the device chosen. Typically the PCI portion of the design will be considerably faster than the input or output device. Because of this it is beyond the scope of this investigation to analyze the data throughput between the on-board FIFO's and an external device. It should be obvious that the *maximum* aggregate bandwidth to an external device would be something less than the maximum achieved across the PCI bus depending on the type of drivers and connection used.

In reality the test software is quite simple. It is responsible for programming the DMA Controller, and then waiting. It does nothing until an interrupt occurs to terminate the cycle.

2.4. System Specification

This section provides an architectural description of the development portion of this thesis. A detailed description of the hardware functionality is provided including block diagrams which show bus organization and data flow paths. A description of the development equipment is provided.

2.4.1. System Architectural Block Diagram

The following diagram describes the overall architecture of a standard PCI Based Personal Computer System. Two systems using this architecture were used to test the final finished prototype boards.

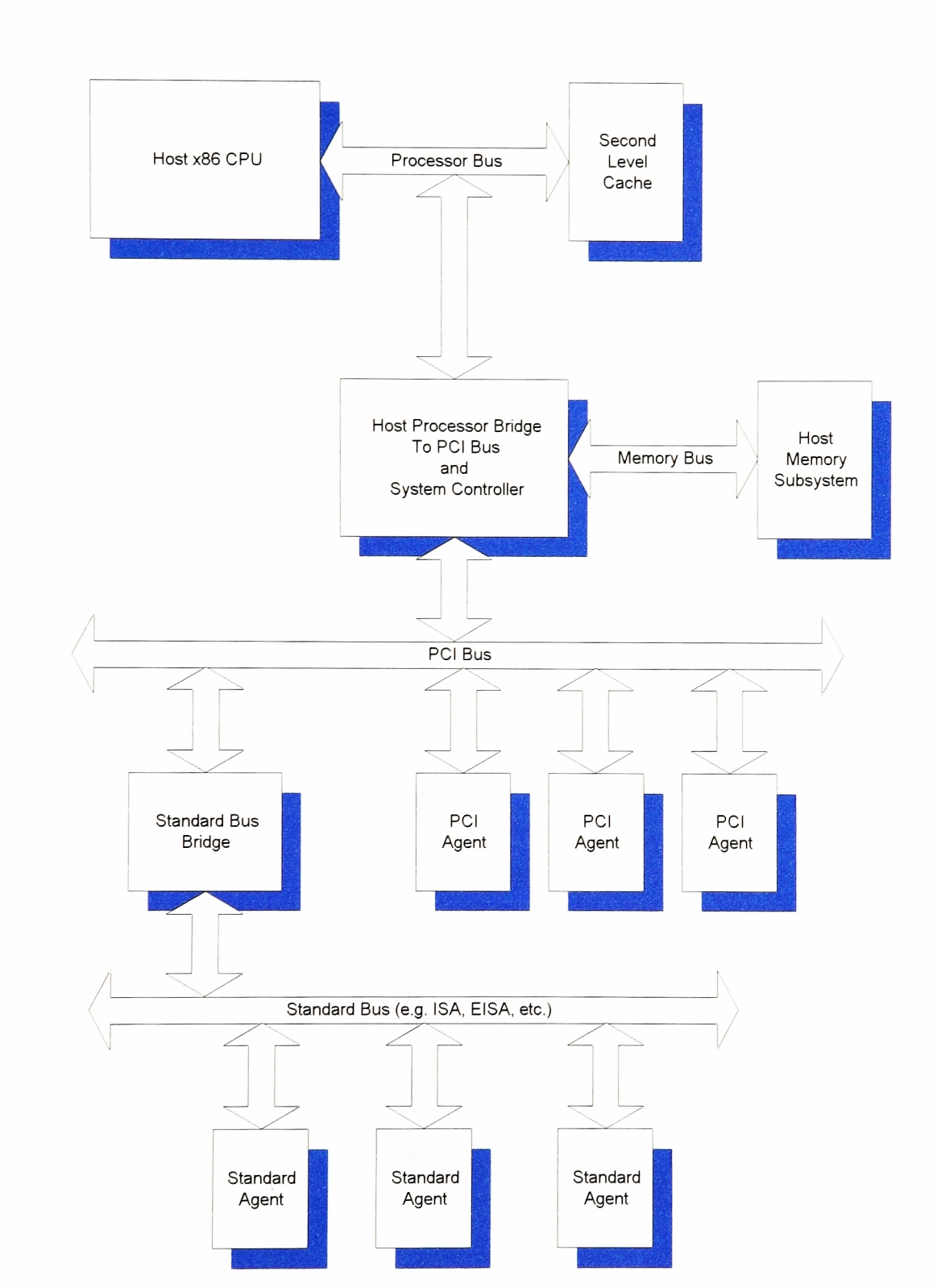
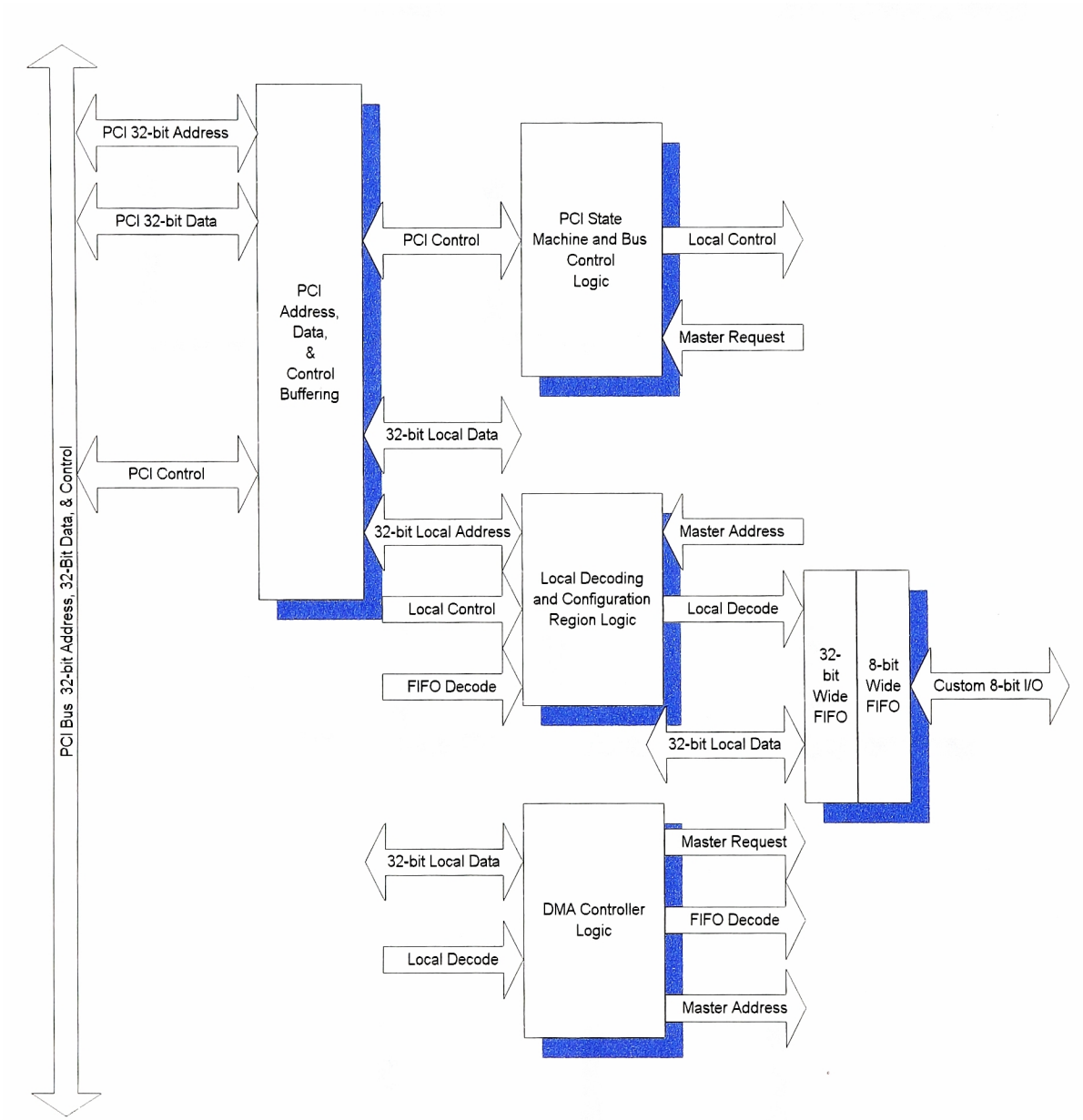


Diagram 2.1: PCI System Architectural Block Diagram

2.4.2. PCI I/O Board Architectural Block Diagram

The following diagram describes the architecture of the proposed PCI I/O board. This block diagram is the first step of the hierarchical top-down design.

Diagram 2.2: PCI I/O Controller Board Block Diagram



2.4.3. Equipment Configuration

Hardware Development and Analysis Tools:
Dataio Dash6 Schematic Entry.
Dataio ABEL6 Logic Development Software
TimingDesigner Professional from Chronology Corporation
Hewlett Packard 1654B Logic Analyzer
Tektronix Oscilloscope

Software Development Tools:
Borland C++ Version 4.2

Performance Analysis System:
Intel Pentium 100 (100MHz Pentium Processor Based PCI Desktop System using 82430NX PCIset Chip-set)
- *Intel Pentium 166 (100MHz Pentium Processor Based PCI Desktop System using 82430FX PCIset Chip-set)*

3. Functional Specification

An overall description and definition of the Development portion of this thesis is provided. Operational and implementation details are described to enable the reader to understand the system that has been developed. A top-down approach is used to describe the overall system with very specific information provided for the most important aspects of the development.

3.1. Hardware Specification

This section provides a detailed look at the operation of the development portion of this thesis. Operation of the PCI I/O board is explained and actual implementation details are provided where required for clarity.

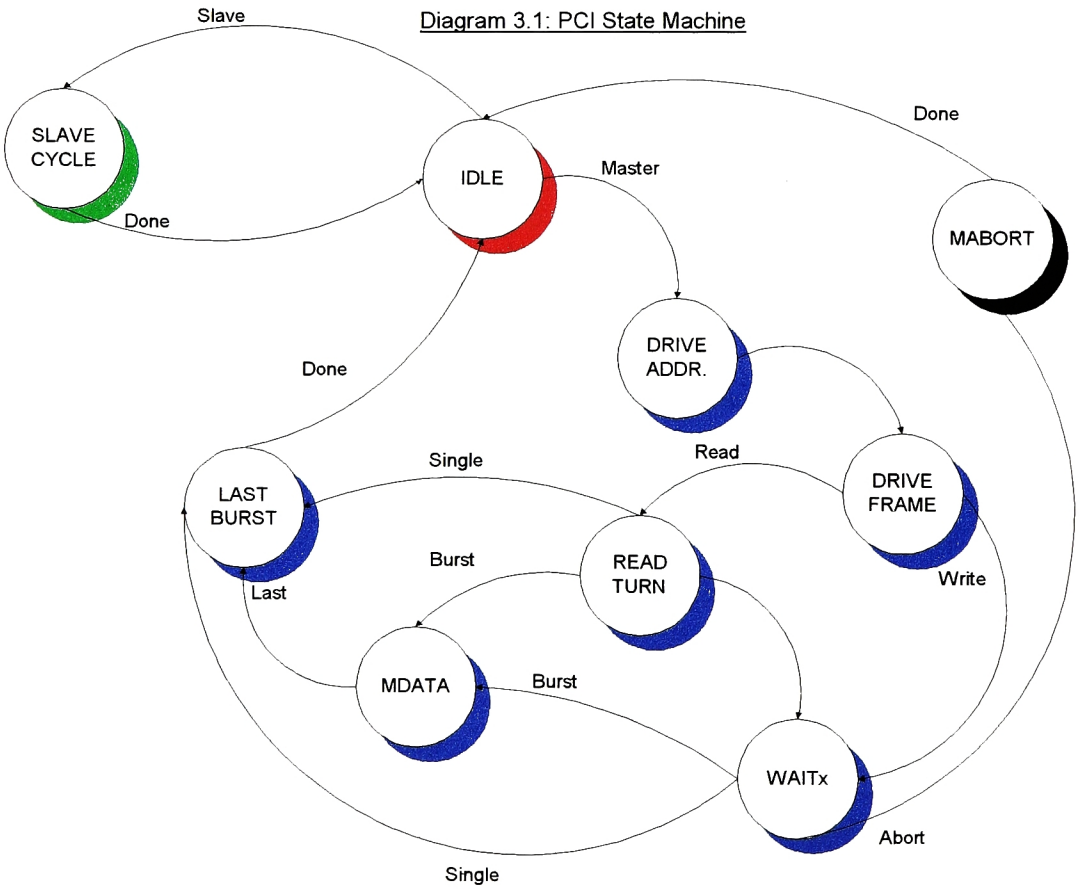
3.1.1. Compatibility

The PCI I/O Board is PCI Bus Compatible as defined within the PCI Local Bus Specification Revision 2.1 dated June 1, 1995 as an Initiator/Target (Master/Slave) Device. Mechanically the Board follows the PCI Local Bus Specification for a 32-bit Standard length expansion card. The only known deviation from this document is that the length of the card will be considerably shorter than a standard full length PCI expansion card.

3.1.2. PCI Interface

The PCI Interface is implemented via multiple Altera EPX780 Field Programmable Gate Array (FPGA) Logic devices. The board is capable of being a PCI Initiator or Target, and is 32-bit PCI compatible. As a Target (Slave) the board is capable of responding to I/O Read, I/O Write, Configuration Read, and Configuration Write Cycles as defined in the PCI Bus Specification. In Initiator (Master) mode the PCI I/O board is capable of performing Memory Read and Memory Write Cycles. It is Initiator mode that is the focus of this board design. Every attempt has been made to make both the Memory Write and Memory Read operations take place as rapidly and efficiently as possible. The unique DMA Controller implementation, which takes full advantage of the PCI Burst Bus functionality, provides a very high bandwidth mechanism for data transfer across the PCI Bus. A state machine technique has been used within the programmable logic to control both Initiator mode cycle generation, and Target mode response. Diagram 3.1 below shows the design of the state machine in a graphical format:

Diagram 3.1: PCI State Machine



3.1.3. DMA Controller

The single channel Fly-by DMA (Direct Memory Access) Controller is implemented with an Altera EPX780 device and a series of discrete counter circuits. The performance goal for the DMA controller design was to be capable of performing 32-bit burst accesses across the PCI Bus in 0-wait states per data phase (maximum theoretical throughput is therefore 132 MBytes/Second). One of the most interesting performance features of this DMA Controller is the burst transfer mechanism. Most DMA Controllers are capable of bursting a fixed number of data elements per address phase. Typical High Speed DMA Controllers are capable of bursting four data elements. The PCI Bus however is not bound by a number of data phases per transaction. The theoretical maximum is infinite. This DMA controller takes full advantage of this bus feature and is capable of bursting up 2^{32} or 4G data elements. This feature is somewhat impractical in today's computers, but the advantages of being able to burst 32 or 64 or even 256 Quad Word Data elements are quite apparent. Much of the overhead for typical bus transactions takes place during the addressing phase of the cycle. If 1 Megabyte of data must be transferred and only 4 bytes of data may be transferred for each address phase generated then 262,100 address phases must be generated in order to transfer the entire megabyte of data. If 128 bytes may be transferred for each address phase only 2048 address phases must be generated to transfer the entire block of data. It is obvious from this simple analysis that the fewer the number of addresses that must be generated the higher the performance if all other factors are equal.

The transaction length of the PCI Bus is bound not by a transfer count, but by a time limit. Each PCI Bus master must implement a Latency timer which is programmed by a systems programmer to provide maximum system performance without causing resource starvation. The system may then be fine tuned for optimum performance. It should be well within the limits of a typical system to be able to burst 32 Quad-Word data elements for a total of 128 bytes per address phase. The DMA controller designed here takes full advantage of that feature by enabling transfer requests to be the entire length of the programmed transfer. It is extremely unlikely that this DMA Controller will ever be the cause for a bottleneck due to burst count latency.

All registers within the DMA controller may be written to and read from by another PCI Master. All DMA transfers take place between system memory and a 32-bit wide on-board First-In-First-Out (FIFO) Memory Buffer. The transfers may be programmed as either Input (FIFO to System DRAM) or Output (System DRAM to FIFO). Memory to memory transfers are not supported in this design. All transfers take place on 32-bit aligned boundaries. Discrete counters are used to control Address generation and transfer counting operations.

The starting address for any DMA Process or Chain Descriptor must be a 32-bit aligned Memory Address; the transfer count is controlled with 32-bit granularity. All DMA Transfers move data between system memory and the on-board FIFO Memory Buffer, and operate in Demand Mode (DMA Requests take place based upon an input signal being driven to the active state by an external device). The PCI I/O board has four 32-bit Read/Write registers are available for controlling and determining status of the DMA Process.

- DMA Source Address
- DMA Transfer Count
- DMA Command/Status
- DMA Next Chain Pointer

The following four sections describe each of these registers in detail.

3.1.3.1. DMA Source Address Register

This 32-bit wide Read/Write register controls the beginning Memory Address of the current PCI Burst DMA Transfer. Prior to initiating a DMA Process this register is programmed with a 32-bit aligned main memory address. This address is the first system memory location accessed during a given DMA Process or Chaining Operation. The completion of a valid data transfer cycle on the PCI Bus is indicated by the signals **IRDY#** and **TRDY#** being sampled active at the same clock edge[3] When the PCI I/O Board is a PCI Master the value in the Source Address Register is incremented to the next 32-bit address for each valid data transfer cycle. The initial value in the DMA Source Address Register is either programmed explicitly by the application in Normal Mode, or it is loaded by the DMA Controller when operating in Chaining Mode.

3.1.3.2. DMA Transfer Count Register

This 32-bit wide Read/Write Register controls the number of 32-bit transfers that take place during a single DMA Process. When the PCI I/O Board is a PCI Master, the completion of a valid data transfer cycle is indicated by the PCI signals **IRDY#** and **TRDY#** being sampled active at the same clock edge. The DMA Transfer Count is decremented for each valid data transfer cycle. The DMA Process is completed when the programmed number of transfer cycles have successfully completed and the Transfer Count Register has reached zero. The DMA Transfer Count Register is either

programmed explicitly by the application in Normal Mode, or it is loaded by the DMA Controller in Chaining Mode.

3.1.3.3. DMA Command/Status Register

This 32-bit wide Read/Write register provides Status information from, and Command information to, the on-board DMA Controller. Table 3.1.3.3 on the following page provides a description of each of the valid bits in the DMA Control/Status Register. The DMA Command Register is either programmed explicitly by the application or it is loaded by the DMA Controller itself in Chaining Mode.

Table 3.1.3.3 DMA Command/Status Register Bit Definition:

Bit Location	Bit Description
Bit 0	**DMA Done** - This read-only status bit is active high upon completion of a Programmed DMA Transfer, and is reset upon system reset; by writing a "0" to the DMA Start Bit, or by writing a non-zero value to the DMA Transfer Count Register.
Bit 1	**INT Pending** - This read-only status bit is active high upon generation of an Interrupt to the Host CPU. It is reset upon writing any data value to the Clear Interrupt Register at offset 6Ch from the Base I/O address.
Bits 7 - 2	Unused
Bit 8	**DMA Start** - This read/write bit controls the initiation of a DMA Process. A Value of 0 indicates that a DMA Process has completed or has been stopped. A Value of 1 indicates that a DMA Process is in progress.
Bits 12,11,10,9	**DMA Cycle Type S0,S1,S2,S3** - These read/write bits control whether the DMA Process is an input (Read from I/O, Write to Memory) or an output (Read from Memory, Write to I/O) as well as the PCI Cycle type that is generated during the transfer: (Bit 12 is S3, Bit 9 is S0). S3 S2 S1 S0 Cycle Type 0 1 1 0 Memory Read 0 1 1 1 Memory Write
Bit 13	Unused
Bit 14	**DMA Mode** - This read/write bit controls Chaining Mode and Memory direction. Mode 0 Count Up, Normal Mode 1 Count Up, Chaining Mode
Bit 15	**Unused**
Bits 17,16	**Interrupt Control Mode 0 and 1** These read/write bits control interrupt generation for DMA Processes either generate no interrupt, Interrupt on Terminal Count (T.C.) or Interrupt at the end of the Last Chain. (Bit 17 = Mode 1, Bit 16 = Mode 0) Mode 1 Mode 0 0 0 No Interrupt Generated 0 1 Interrupt on T.C. 1 0 Interrupt at End of Last Chain 1 1 No Interrupt Generated
Bits 18-31	Unused.

3.1.3.4. DMA Next Chain Address Pointer Register

This 32-bit wide Read/Write register specifies the address of the next Chain Descriptor to be accessed when the DMA Controller is in Chaining Mode. This register provides the link between descriptors during scatter or gather type operations. When the DMA Controller is initially programmed in Chaining Mode this register is programmed with the value of the location of the first Chain in the programming link. Subsequent Chaining information is loaded without programmer intervention by the DMA Controller once the current chain has been completed. A value of 00000000h indicates the final Chain in the series. The format for a Chain Descriptor in Memory is shown below.

Table 3.1.3.4 Chain Descriptor Format:

Chaining Descriptor Information(32-bits Wide)	Address
Control Word	n
Transfer Count	n + 4
Next Chain Address	n + 8
Start Address	n + 12

The memory locations for a single Descriptor must reside in four 32-bit aligned 32-bit wide contiguous memory locations. Individual Descriptors may be located anywhere within the 4 Gigabyte System Memory Space as long as they are 32-bit aligned. The 32-bit Next Chain Address values represent actual physical memory locations within the System Memory Space.

3.1.4. On-board Data Flow Control

The on-board DMA Controller operates in a Demand Mode fashion. When properly programmed, DMA Operation commences upon the receipt of a DMA Request signal from the Parallel Interface Control logic. This DMA Request is based upon a transfer being initiated, and data (or space depending upon direction of transfer) being available in the on-board FIFO Memory Buffer. There are two types of DMA requests that may be generated by the I/O Interface Logic. During normal operation a Burst Request is performed by the Parallel Interface Control Logic. This causes the DMA Controller to attempt to perform data transfers until all programmed DMA transactions have been completed. As long as there is a transfer request from the DMA Controller to the PCI Bus Interface Logic, the PCI Bus interface logic will request the PCI bus according to the rules described in the PCI Specification. Once a PCI transaction has been initiated, transfers will continue until one of the following events occurs:

a) The **GNT#** signal on the PCI Bus has been removed by the PC Host's PCI Arbiter *and* the local on-board **latency timer** has expired.
b) The **STOP#** signal has been asserted by the slave device being accessed on the PCI Bus indicating that it's transfer buffer's are full in input mode, or empty in output mode.

The PCI Interface logic will remove it's bus request signal (**BREQ#**) and re-initiate the arbitration process by re-asserting **BREQ#** upon the occurrence of one of these two events.

In Input Mode, when **PAGE_SYNC** is active and the FIFO status is "Almost Full", DMA Burst requests occur. The same is true in Output Mode when the FIFO status is "Almost Empty" During Input Mode, once **PAGE_SYNC** is removed by the external Input logic, the DMA requests will be non-burst. This causes the PCI Transfers to contain a single data phase until the FIFO Memory Buffer has been completely emptied. During this non-burst mode there will be a single data phase for each PCI Transaction. This logic is in place to prevent an overrun or underrun condition from occurring in the FIFO's.

3.1.5. PCI Configuration Region

The PCI Configuration Region is implemented via an Altera EPX780 FPGA Device. A 256 byte register space organized as sixty-four 32-bit words is implemented to perform the configuration functionality. The PCI Configuration Region is required in all PCI compliant devices. The register set associated with this region is configured at system initialization for proper operation in the host system. Since this region is standard PCI fare it will not be discussed in detail during this thesis, and is considered beyond the scope of this investigation.

3.1.6. Parallel Data Interface

The Parallel Data Interface is an 8-bit Data, Control, and Status interface for communication between the on-board FIFO memory buffer and the off-board I/O Device. This is a general purpose interface used for demonstration purposes only. In an actual application this logic, which is contained in a CPLD, would be designed to communicate with a specific device or set of devices. The on-board control registers are used to program the interface in either the input or output mode. In input mode data is transferred from an input device to the FIFO based upon the status of Input device and the FIFO (i.e. Does the input device have data to send, and does the FIFO have room in which to store the data being sent.). In

the output mode data is transferred from the FIFO to an output device. Again, the status signals are used to condition this transfer.

In the CPLD Source file provided with this thesis there are fourteen pins available for implementing a custom bi-directional interface. This flexibility enables the designer to connect the PCI I/O board to a wide range of devices without any modification to the PCI interface logic.

3.1.7. JTAG Interface

A JTAG (Joint Test Action Group, IEEE 1149.1) interface is implemented in order to enable on-board programming of the SRAM Logic Array's in the Altera Flexlogic devices during system debug. A standard 20-pin Insulation Displacement Connector (IDC) header is used to interface the PCI I/O Board to a PC parallel port based JTAG programming interface for the Flexlogic.

3.2. *Software Specification*

3.2.1. Compatibility

The software developed for this thesis executes in a DOS 5.0 or greater environment. Protected mode drivers are utilized in order to read and write memory above the 1 Megabyte DOS boundary in order to test operation of the DMA process. The software is not intended to be a robust end-user application, but rather, to demonstrate the functionality and performance of the PCI DMA subsystem.

3.2.2. PCI Plug'n Play Usage

The PCI I/O Board is fully PCI compliant and supports the entire required Configuration Region. Because of this, the diagnostic software is able to determine where the board is located within the system's I/O Map and perform accesses to it accordingly. Unlike ISA or EISA type add-in cards PCI Boards require no I/O location, Memory location, or interrupt jumpers.

3.2.3. DMA Controller Programming

3.2.3.1. Programming the DMAC in Normal Mode

When initializing the DMA Controller for Normal (non-chaining) Mode the first programming operation is to write bit 8 of the DMA Control Register (DMA Start) with a "0" This ensures that the DMA Controller is in a halted state. The Source Address and Transfer Count Registers may then be programmed in any order.

Finally the DMA Controller should be written with the appropriate values for the desired mode, as described in Table 3.1.3.3, with the DMA Start Bit set to a "1" to initiate the DMA Process.

Writing a "1" to the Start Bit of the DMA Controller causes the DMA Controller to initiate a DMA Transfer by requesting the PCI Bus. Once the bus has been granted the DMA Controller will drive the current address onto the PCI Bus along with the appropriate PCI Command signals to indicate the type of transfer as specified in Table 3.1.3.3. Once initiated the Normal Mode DMA Process will continue until one of the following occurs:

- The Transfer Count Register has reached a value of zero.

- The DMA Start bit in the DMA Control Register is written with a Zero (Process will be suspended at the completion of the current transaction, and may be restarted by rewriting the DMA Start bit with a One).

3.2.3.2. Programming the DMAC in Chaining Mode

When initializing the DMA Controller for Chaining Mode, the first programming operation is to write bit 8 of the DMA Control Register (DMA Start) with a "0" This ensures that the DMA Controller is in a halted state. Prior to initiating the DMA Process by programming the DMA Control Register for the desired chaining operation, the Chaining Descriptors should be programmed in System Memory. Each chaining descriptor ***must*** be located at a 32-bit aligned address. This enables the DMA Controller to properly fetch the chaining information. The chaining descriptors do not need to be located in a contiguous block of memory. Each descriptor may exist anywhere in memory as long as the address is 32-bit aligned. The Next Chain Address Pointer Register within the DMAC should be programmed after the chaining descriptors have been loaded into system memory. The value of this register should be the base address of the first chaining descriptor in System Memory. The Source Address and Transfer Count Registers are don't care values for the Chaining Mode initialization process. They will be loaded by the DMA controller when the first chaining descriptor is accessed. Programming the Next Address Pointer Register enables the DMA Controller to access the first Chain Descriptor in System Memory. Once the Next Address Pointer Register and Chaining Descriptors have been established the DMA Control Register may now be programmed to

operate in Chaining Mode according to the description in Table 3.1.3.3, and the DMA Start bit may be set to a "1" to initiate the DMA Process.

Once initiated, the Chaining Mode Process will continue until the Transfer Count has reached zero *and* the Next Address Pointer has been loaded with null descriptor reference (00000000h). As presented in the Control Register Bit Definition, an interrupt may be generated at the end of each chain (each time Terminal Count Reaches Zero), at the end of the last chain (Terminal Count is Zero and Next Address Pointer is Zero), or not at all. This may be changed on a chain by chain basis as the Control Word is the first value loaded during the chaining process.

4. Analysis

Performance data acquired during testing is presented and analyzed. Two measurement techniques were invoked in order ensure the data obtain was accurate. Two popular PCI systems were used to host the PCI I/O board for these performance measurements.

4.1. Performance Measurement

One of the main goals during the development of this thesis was data movement performance. This system architecture was defined to enable the DMA Controller to operate with minimal CPU overhead. This accomplishes two goals:

1) The CPU is freed up to perform other tasks
2) The overall DMA task becomes more efficient

Data throughput was performed in two different types of Personal Computer Systems. This was done to demonstrate that overall performance is greatly dependent upon the performance of the PCI system in which the PCI I/O board resides. Additionally, two independent measurement techniques were used in order to validate DMA transfer performance. One method invoked a logic analyzer to time the duration of a given transfer by sampling signals specific to the beginning and end the process. The second technique involved writing software to utilize the programmable interval timer on the PC and calculate the bandwidth of a transfer. This approach provided two separate and independent measurements for a series of complete DMA transactions

In addition to the diagnostic equipment used to measure performance, the board design was slightly modified to more accurately determine the maximum aggregate bandwidth of the system. The obvious bottleneck in the overall system is the ability of an external device to either fill or empty the on-board FIFO memory. The FIFO memory is meant to be used as a buffer to alleviate bandwidth discrepancies between the PCI Bus and the external resource. The goal of the

performance measurement is to determine what type of sustained throughput capability this board could have in a lightly loaded system. To achieve this end the DMA request input to the DMA Controller was modified for testing purposes. During normal operation the DMA request input is used to signal the DMA Controller that the FIFO is able to accept (in output mode) or provide more data. Even for a relatively fast external system the DMA Controller would not typically be able to attempt to burst more than 4 to 6 K Bytes without having to wait for the external device to move more data. In order to simply test the throughput of this PCI board the DMA request input was modified to be always requesting a DMA Transfer. In Output Mode this would mean that the FIFO would always appear to have space available for more data, and in input mode this would mean that the FIFO would always appear to have more data to provide. None of the cycle timing was in any way modified to accomplish this. All read and write cycles to the FIFO were performed to ensure that no performance enhancement occurred as a result of this modification. This enabled much larger blocks of data to be transferred without having to wait for an external device to process data. This enabled testing to be performed with data transfer sizes limited only by the amount of memory in the host PC System.

4.1.1. Logic Analyzer Measurement

A Hewlett Packard 1654B Logic Analyzer configured in state mode was used to capture signals pertinent to the beginning and end of a DMA Transfer Operation. The goal of this type of measurement is to accurately select an event that occurs immediately prior to the DMA process initiating, and then another event that occurs immediately after the transfer has been completed.

For measuring throughput the signal DMA_REG2_RD# was selected as a trigger mechanism to enable the logic analyzer to generate time stamps for both the beginning and end of the DMA transfer. This is the write strobe for the DMA Control Register which is written once to initiate the DMA operation, and then again, immediately subsequent to the DMA operation completing, to clear the DMA_START bit within the register. These two write operations serve to envelope the entire programmed DMA operation providing an analyzer reading of total elapsed time for the DMA transfer. Throughput, in bytes per second, is then calculated by dividing the total number of bytes transferred by this elapsed time in seconds. The logic analyzer has a 4ns granularity which is more than sufficient to provide reasonable accuracy for these throughput measurements. This technique was used to determine all measurements listed under the heading of Logic Analyzer.

4.1.2. Programmable Interval Timer Measurement

In order to provide correlation with the hardware measurements described above, the test routine included a mechanism to provide automatic

performance measurement information. The System Timer on the PC Host was used to enable the software to acquire accurate time-stamp information at the beginning and end of each block of data transferred.

The PC's Timer 0 is a programmable counter with an input frequency of 1.191318 MHz. The timer is typically configured to operate at an output frequency of 18.20 Hz when the PC is powered up[4] Immediately prior to setting the START bit to initiate the programmed DMA operation the on-board 16-bit timer, Timer 0, is interrogated for use as the initial time stamp. Once the programmed DMA transfer is finished, a completion notification interrupt is generated to the host CPU. Immediately subsequent to this Interrupt being serviced, Timer 0 is once again interrogated to determine the end time of the transfer. For those transactions which caused an overlap in the counter (i.e. the counter decrements to zero and automatically starts counting at 0xFFFF again) an Interrupt Service Routine (ISR) was installed to count each of these situations. Therefore the algorithm used for determining the actual number of timer ticks was implemented as follows:

Assuming the count parameters of:

Initial count = i
Final Count = f
Interrupt Count = n
Timer Ticks = t;

if(n = 0) then
t = i - f
else
t = (n-1) * (0xFFFF) + (i + (0xFFFF - f))

Once found, t is multiplied by the period of 1 timer tick or 838.0965 x 10^{-9} to acquire the total elapsed time. Throughput, in bytes per second, is then acquired by dividing the total number of bytes transferred by this elapsed time in seconds. This technique was used to determine all measurements listed under the heading of Software Timer.

4.1.3. Performance Data

Two separate off-the-shelf PC systems were used to benchmark the transfer performance of the PCI I/O Board. Both systems included a 32-bit wide 33Mhz PCI bus. The first system, referred to as the Pentium 100, was a 100Mhz Pentium based platform using the Intel 82430NX PCI Chip-set. The System Controller in this Chip-set provides two 4-DWORD PCI-to-Memory Posted Write Buffers and one 4 QWORD PCI-to-Memory Read Pre-fetch Buffer[5]. The buffering is used to enhance PCI Burst performance by not requiring the PCI Initiator to wait for accesses to

memory to take place. The second System, referred to as the Pentium 166, was a 166Mhz Pentium based machine using the Intel 82430FX PCIset. This Chip-set provides 12-DWORDs of PCI-to-Memory write posting and a snoop-ahead feature so that PCI-to-memory bursts in either direction can be sustained at up to 120MBytes/Second[6] The performance differences between platforms are not indicative of the processor speed, but are the result of an improved buffering and DRAM interface scheme within the PCI Chip-set used in the 166Mhz platform. It is important to understand that the CPU is not involved in the data transfer process, other than initially programming the registers of the DMA Controller. Additionally, performance measurements are not initiated until the CPU has finished programming the DMA operation. As the PCI bus has become more prolific the designers have continually improved the performance of the Chip-sets. It is also interesting to note the discrepancy between PCI read and write performance especially in the Pentium 100 System. During a write operation the data is present at the Initiator and is therefore ready and valid very early in the cycle. The initiator does not have to wait for the actual write to DRAM to take place; only for the data to be latched into the Chip-set's buffer. A read operation, however, requires the data to be extracted from the DRAM subsystem prior to completing the data transfer across the PCI bus. This requires the initiator to wait for the complete DRAM access prior to receiving data.

The following two pages include tables which provide data size and throughput information for PCI reads and writes for each of the host systems utilized for this analysis:

Table 4.1 PCI Read Performance Data

Transfer Size (Bytes)	Pent100 Software Timer Rate (MB/S)	Pent100 Logic Analyzer Rate (MB/S)	Pent166 Software Timer Rate (MB/S)	Pent166 Logic Analyzer Rate (MB/S)
32	1.92	1.93	2.6	2.63
64	3.31	3.02	4.85	5.04
128	4.41	4.55	8.57	8.98
256	6.20	5.95	15.33	15.45
512	7.57	7.32	22.40	22.44
1,024	8.03	8.12	30.66	30.33
2,048	8.60	8.62	36.41	36.63
4,096	8.93	8.90	40.17	40.45
8,192	9.05	9.04	43.15	43.07
16,384	9.10	9.11	44.49	44.52
32,768	9.14	9.14	46.31	46.37
65,536	9.15	9.16	47.52	47.53
131,072	9.18	9.18	48.12	48.15
262,144	9.18	9.18	48.41	48.44
524,288	9.15	9.16	48.57	48.59
1,048,576	9.18	9.18	48.63	48.66
2,097,152	9.17	9.17	48.67	48.69
4,194,304	9.17	9.18	48.69	48.71
8,388,608	9.17	9.18	48.69	48.72
12,000,000	9.17	9.18	48.69	48.72

Table 4.1 clearly shows the increased read performance of the 82430FX Chip-set demonstrated by the Pentium166 system. It is also very interesting to note the relationship between transfer size and aggregate throughput. This trend is present for all transfers analyzed. It is important to realize that the CPU programming overhead is not present in any of these numbers. The time required for the CPU to program the DMA registers is not measured, and therefore does not skew the results in the favor of the longer transfer sizes.

Table 4.2 PCI Write Performance Data

Transfer Size (Bytes)	Pent100 Software Timer Rate (MB/S)	Pent100 Logic Analyzer Rate (MB/S)	Pent166 Software Timer Rate (MB/S)	Pent166 Logic Analyzer Rate (MB/S)
32	2.14	2.20	3.10	3.26
64	4.55	4.03	6.05	6.20
128	7.28	7.02	11.27	11.30
256	10.79	11.14	18.20	18.17
512	16.64	15.67	26.48	26.08
1,024	19.75	19.78	35.30	35.59
2,048	23.07	22.44	42.37	42.79
4,096	24.66	24.55	48.04	48.01
8,192	25.61	25.54	50.93	51.06
16,384	26.14	26.09	52.66	52.73
32,768	26.40	26.37	53.49	53.57
65,536	26.50	26.49	53.99	54.02
131,072	26.56	26.58	54.21	54.23
262,144	26.61	26.62	54.31	54.35
524,288	26.44	26.46	54.37	54.37
1,048,576	26.63	26.65	54.40	54.44
2,097,152	26.64	26.65	54.41	54.44
4,194,304	26.64	26.65	54.42	54.44
8,388,608	26.64	26.65	54.41	54.42
12,000,000	26.64	26.66	54.41	54.44

Again the Pentium166's 82430FX Chip-set provides a significant performance benefit over the 82430NX Chip-set used in the Pentium100 system. Additionally there is very good correlation between the two measurement techniques implemented for both systems at all performance levels. It is also interesting to see the significant performance improvement realized by the Pentium 166 system. The write performance is greater than twice that of the Pentium 100 system and the read performance is greater than five times better. The following four traces captured on the Hewlett Packard Logic Analyzer demonstrate why this performance discrepancy was realized:

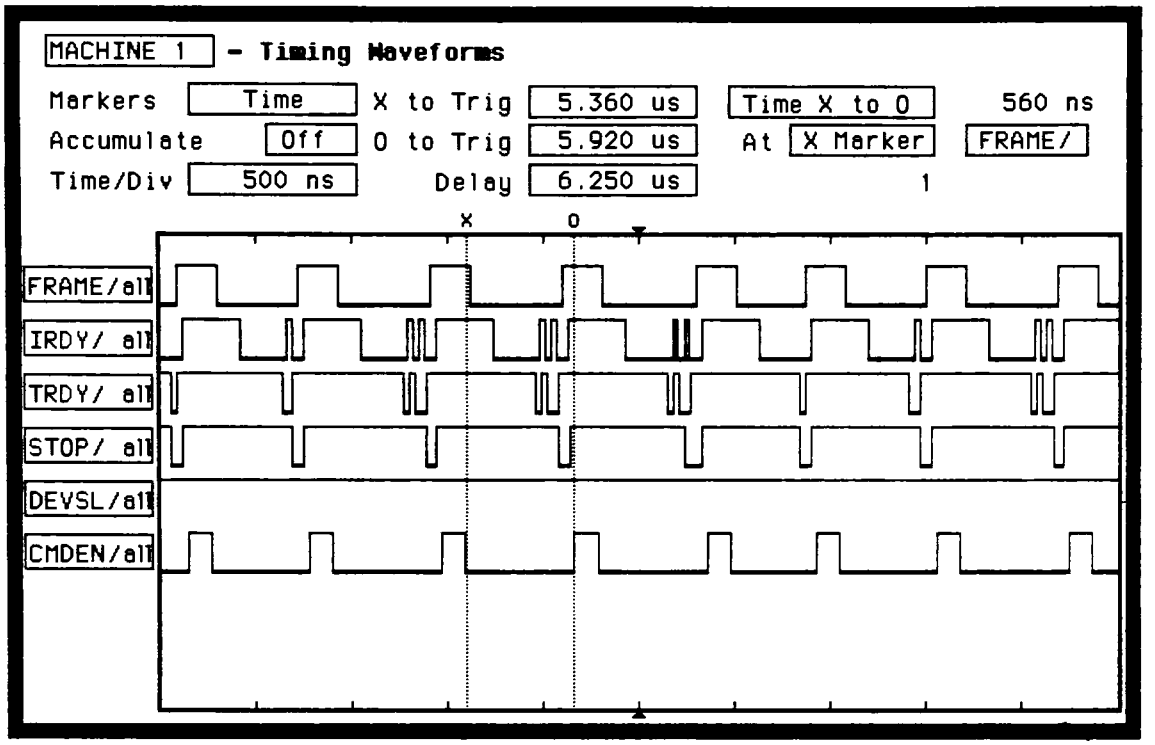


Diagram 4.1 PCI Read Pentium 100 System Logic Analyzer Trace

Diagram 4.1 shows that the Pentium 100 system used does not handle PCI reads very effectively. There is a delay of approximately 300ns before the NX Chip-set returns TRDY/ active indicating that it is able to provide the requested data. Additionally a maximum of three data phases (and sometimes only one) take place for each assertion of the signal FRAME/ on the Pentium 100 System. The reason for this is that every transaction ends with the active low assertion of the STOP/ signal by the Host Chip-set. This indicates that the motherboard is no longer able to provide data for the current request in a timely fashion. The STOP/ signal is used to notify the Initiator (in this case the PCI I/O Board) that it should re-attempt the request later. It is essentially a negative acknowledge. Because of STOP/ being driven the PCI Board only has control of the PCI bus for about 560ns each time it is granted the bus. This trend continues for the duration of the transfer and is the main reason the transfer rate for PCI Reads in the Pentium 100 System are relatively low performance compared to the Pentium 166.

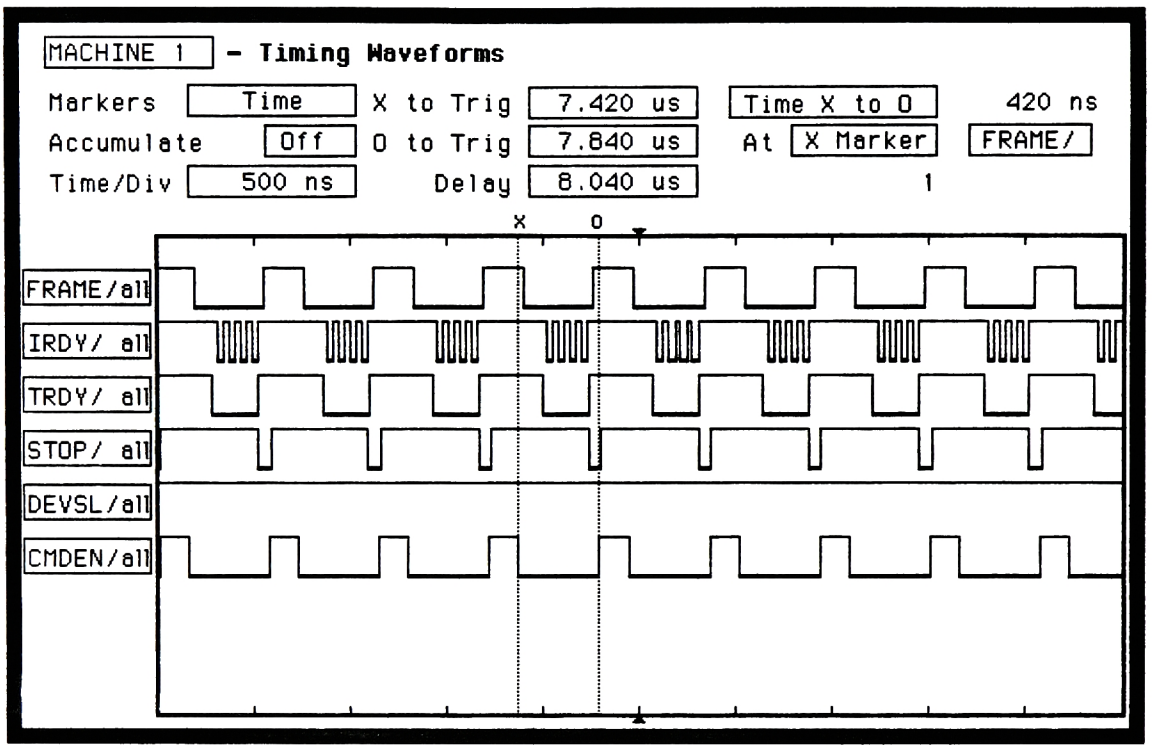


Diagram 4.2 PCI Write Pentium 100 System Logic Analyzer Trace

Diagram 4.2 shows much better performance characteristics than the previous example. In this case the Host system returns TRDY/ active low indicating it is ready to accept the data very early in the cycle. Additionally this signal is held low for the duration of the cycle until STOP/ is asserted. This illustrates the improved performance that is acquired through write-posting. The internal buffering of the NX Chip-set is able to accept four 32-bit values before it is filled and must wait for the data to actually be transferred to the DRAM subsystem. The FIFO can be filled by the high-speed PCI accesses thus efficiently using the bus. The data can then be written to DRAM at the speed of the Memory subsystem without impacting other devices which may want to utilize the PCI bus. In this case, for each sequence of four data phases, the Host is not slowing the transaction. The PCI Board, which must negate IRDY/ each time a data transfer occurs, is causing the FIFO filling process to be approximately twice as long as would be required by the Host. The current request, however, is stopped each time, by the Host Target, and must be re-attempted by the Initiator. The PCI Board only has control of the bus for approximately 420ns, but is able to transfer 16 bytes of data in that time.

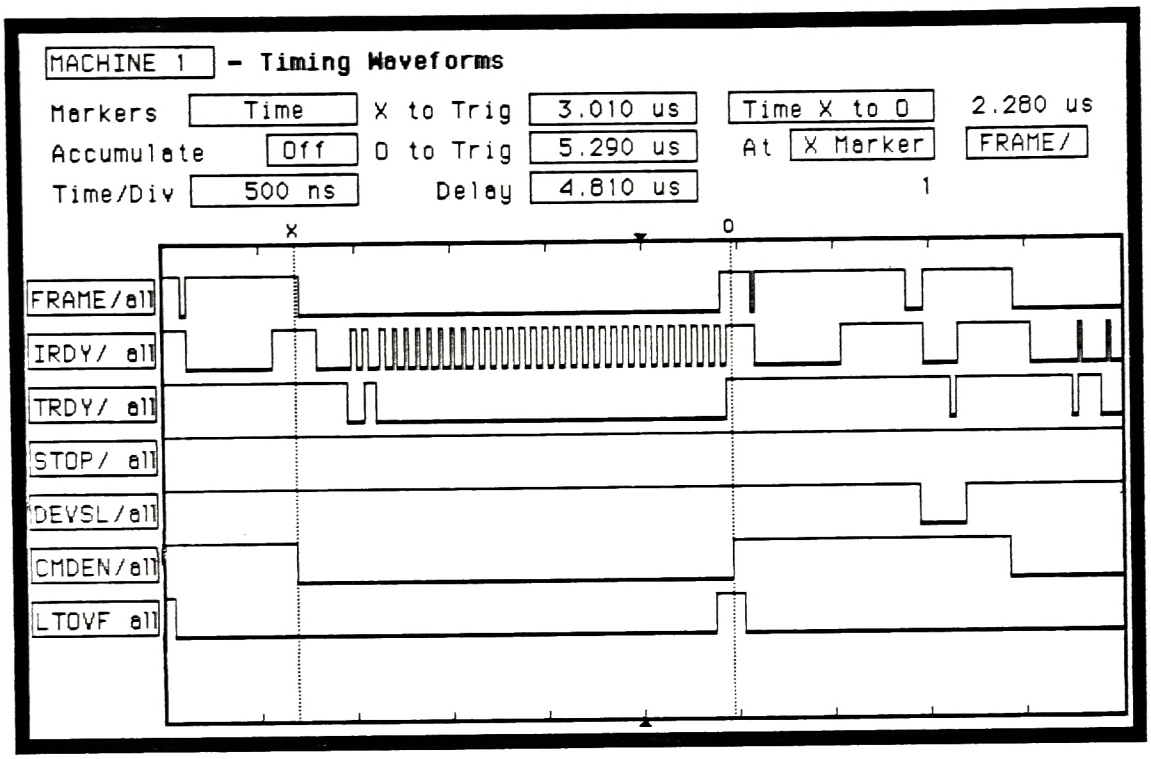


Diagram 4.3 PCI Read Pentium 166 System Logic Analyzer Trace

Diagram 4.3 shows a marked improvement in performance over either of the traces acquired from the Pentium 100 system. The initial delay to TRDY/ being driven is approximately half of that seen in Diagram 4.1. Even more significant is the fact that STOP/ is not driven at all during these transfers. This transfer is terminated because the PCI I/O Board's Latency Timer has expired indicating that it should relinquish control of the bus for overall system performance reasons. In this example 33 data phases take place in a single transaction. The PCI Board is clearly the performance limiting device here. This indicates that in FX Chip-set and the associated DRAM subsystem are able to keep up with the PCI Board over relatively long spans of time. The PCI Board maintains control of the bus for approximately 2.28uS and transfers over 10 times as much data as it was able to in the Pentium 100 system. This is due to the fact that the FX Chip-set has the ability to pre-fetch read data if it anticipates that it may be needed for subsequent accesses by a PCI Initiator. Therefore data is actually read from DRAM before it is *needed in* order to satisfy the timing requirements of high performance systems.

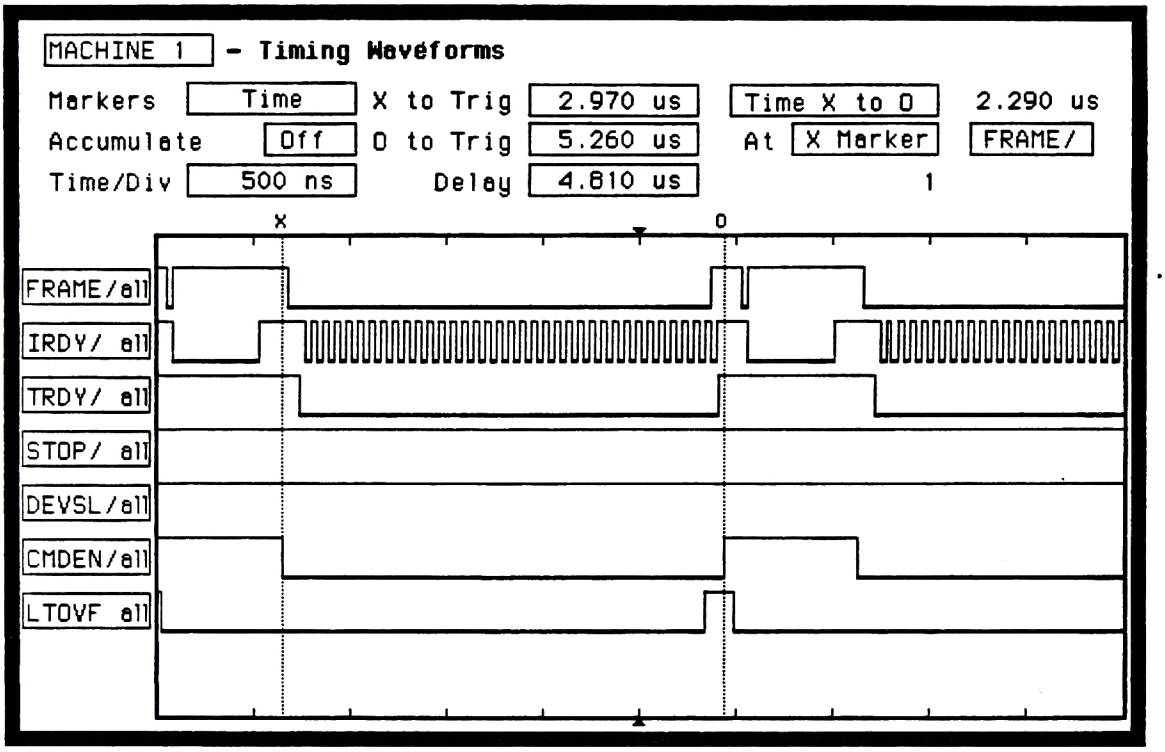
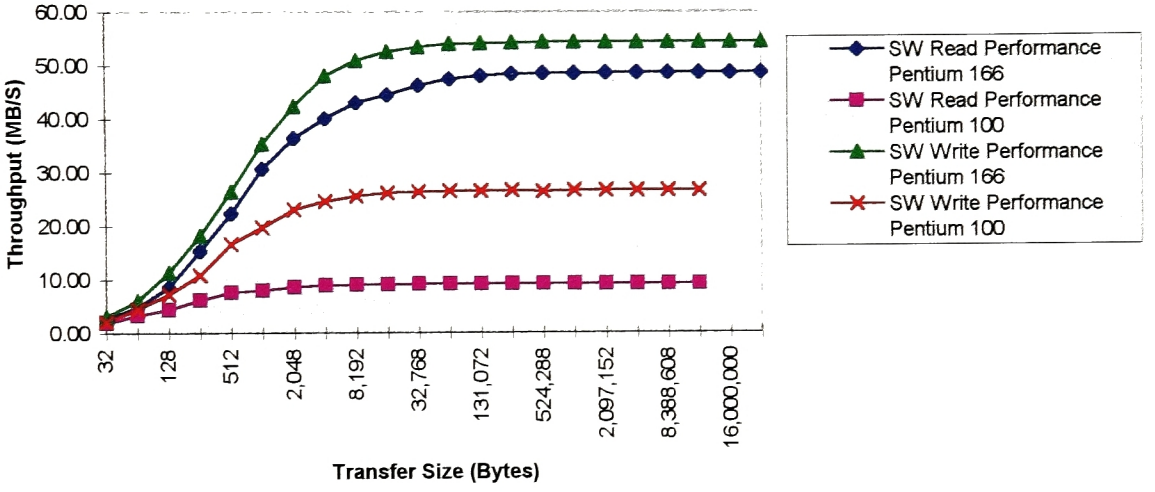


Diagram 4.4 PCI Write Pentium 166 System Logic Analyzer Trace

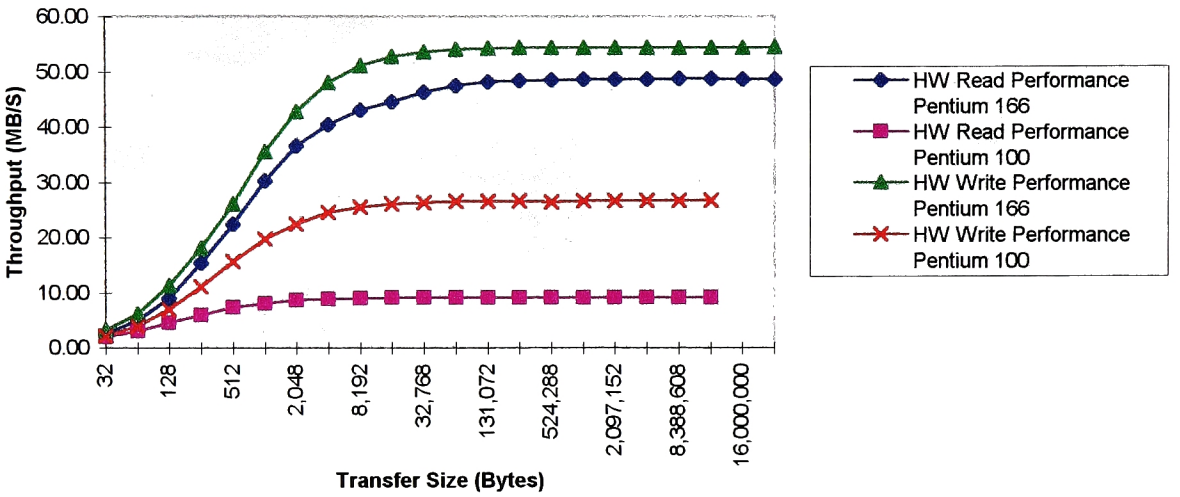
Diagram 4.4 shows the fastest transfer investigated. PCI write throughput was measured at a maximum of more than 54 Megabytes per Second. In this example 36 PCI write transfers occur during a single bus transaction. The reason that this performance is slightly higher than in the Read direction is the delay to the initial TRDY/ being asserted. Because this is a write, and no pre-fetching is required, the PCI Board does not have to wait for data from the Host. The write-posting buffers are filled immediately. Once again it is the PCI Board's Latency Timer that expires causing the cycle to be terminated.

The following two graphs demonstrate not only the close correlation between the two performance measurement techniques, but also magnify the relationship between transfer size and data throughput for each of the two systems analyzed:

Graph 4.1: Software Timer DMA Performance



Graph 4.2: Hardware DMA Performance



Because of the close correlation between the two performance measurement techniques it is somewhat difficult to see any difference in the data if plotted according to system. Therefore, the data is provided in graphical format according to method of measurement as shown in the two graphs above. It is quite clear that as transfer size increases, performances increases until an asymptote is reached. This demonstrates the advantages of the increased burst count capability of the DMA Controller designed for this thesis. Many standard off-the-shelf DMA controllers are capable of transferring only 16 bytes per *burst* transaction. At that rate less than 10% of the capability of the 82430NX PCIset used in the Pentium 166 system is realized. Even without the ability to transfer data at the maximum theoretical bandwidth of the bus a significant performance increase was realized by the increased burst size capability of this DMA Controller.

4.2. Adherence to Goals and Problems Encountered

In general this design was extremely successful, and met the vast majority of design goals. The design is versatile, reliable, has a straightforward programming interface, and is very high in performance.

4.2.1. Goals Achieved

The two most significant goals were the ability to burst in an essentially unlimited fashion and the ability to transfer one 32-bit wide piece of data on every clock. The first of these two goals was met and the second was not. A 15.2ns timing margin error resulted in a transfer data reduction from the maximum PCI bus throughput rate of 132 Megabytes per second to 66 Megabytes per second. Even with this reduction in throughput the PCI I/O board performed commendably in both systems analyzed. Based upon the results acquired it would be preferable to sacrifice the transfer rate over the burst transfer size. The data indicates that there is a 5x increase in performance for larger size transfers. Doubling the transfer rate to the design goal would likely not have produced twice the performance for small transfers in the Pentium 100 system due to the fact that buffers would have filled faster. In the Pentium 166 system a doubling effect may have been achieved, but that would have only provided a performance of approximately 5 MBytes/Second for 32 byte transfers. An acceptable performance compromise might seem to be to provide a 16-bit counter, which would enable the transfer of 256Kbytes of data, if the ability to transfer at 132MBytes/Second were present. This however would provide limitations to the image size that could be transferred with a single programmed transfer, and may render the PCI I/O board unacceptable for certain applications.

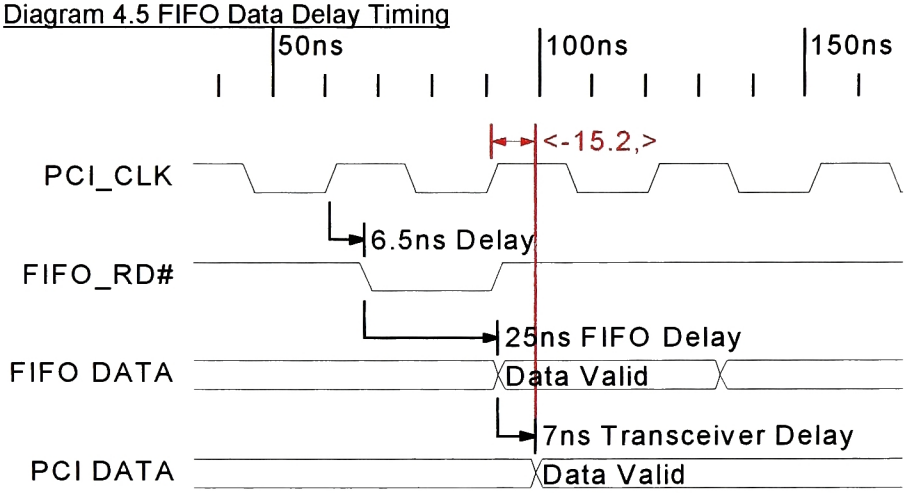
The secondary goals of reliability, reusability and a robust programming interface were all accomplished. The design of the PCI I/O board was successfully partitioned to enable a designer to easily transition this design into a custom interface solution. The PARIO PLD shown in the appendix includes fourteen control signals which may be used to implement a robust solution. The programming interface for the PCI I/O board is extremely straightforward. By simply programming the source and destination registers, the transfer count register, and the command register a DMA transfer is initiated. Completion can be detected by either an interrupt or polling mechanism. Scatter-gather or chained DMA operations are also easily understood.

4.2.2. Transfer Rate Problem

Reducing the data throughput from one data transfer on *every* clock, to one data transfer *every other* clock, was necessary due to a timing problem found when the PCI I/O board was operating in input mode (Performing PCI Write Operations). This was due to the read data delay time of the FIFO. The fastest FIFO device which met the overall functional requirements that could be found at design time was a Texas Instruments SN74ACT2235-20. This 1024 x 9 x 2 device has a read timing delay of

25ns. This, coupled with a propagation delay of 7ns through the IDT74FCT16646CT Registered Transceivers which are connected to the PCI Bus created an overall delay in the data pipe of 38.5ns as shown in the following timing diagram:

Diagram 4.5 FIFO Data Delay Timing



A PCI Bus operating at 33MHz requires the worst-case data delay be limited to 23.3 ns from the rising edge of the clock. This conflict was only present when performing PCI Writes, and was not noticed until the board was manufactured and being tested. PCI Read operations could still be performed at the desired 132Mbytes per second. Because it was intended that this board be production-worthy, it was decided that it made more sense from a marketing standpoint to operate the board at the same rate in both directions. This meant operating at a maximum sustainable rate of 66Mbytes per second in both directions.

4.2.3. Transfer Count Problem

The on-board Transfer Counter/Register is 32-bits wide and can be used to count transfer's up to 4Gigabytes in length. Whenever a transfer has been initiated and the Transfer Count Register has not expired, the PCI I/O Board is ready to perform a transfer, and will continue to do so until it is removed from the bus. There was a significant problem which occurred in

this area of the design. A series of four 8-bit high-speed counters are daisy-chained to provide a full 32-bit count. . In the original design only the output of the final counter stage was sampled by the logic to determine if the transfer count had expired. This configuration was intended to operate as a synchronous counter using the 33Mhz system clock This created a problem due to the propagation delay of the counter devices themselves. The maximum propagation delay from the Count Enable input to the Terminal Count Output for the 74F779 device is 8ns. Four of these delay's is longer than the minimum period of the system clock. The problem that resulted was that one extra transfer would take place depending upon the number programmed into this Transfer Count Register. If the upper register was utilized in the count size an extra data phase would occur. The solution to this problem was to re-synchronize the output of the second stage of the counter with the 33Mhz system clock. This guarantees adequate setup and hold time for sampling of both the second-stage re-synchronization, and the final Terminal Count output signal DMA_DONE#. The one wrinkle that this hardware modification introduced was that the DMA Controller operation was altered to perform one additional transfer after the transfer count register expires. This can be worked around in software by simply programming the transfer count register with one less than the actual number of transfers desired. This is a common technique used in counting type devices such as DMA Controllers or synchronous counters.

4.3. Potential Improvements and follow-up work

Much was learned during the development and analysis of the PCI I/O board and DMA controller. As with any project of this magnitude and complexity, upon completion, areas of improvement can be clearly defined. It is my opinion that this work could be improved upon given similar development constraints from a performance standpoint. When this project was initiated there were no CPLD's which were fully compliant with the PCI Specification from both a drive capability, and a capacitive loading standpoint. Rather than violate that specification the decision was made to utilize buffers between the PCI bus and the on-board logic. This decision ultimately lead to the requirement to turn IRDY# off for one clock after every successful data transaction which reduced the maximum potential throughput to 66Mbytes per second. Devices are now available which adhere to the PCI specification and could be used to make this a simpler and more elegant design from a bus interface standpoint. In addition, the usage of these compliant components along with faster FIFO's should lead to the ability to transfer data on *every* clock instead of on *every other* clock, thus doubling the performance.

This analysis was performed by focusing on the PCI bus bandwidth. Every effort was made to successfully transfer data across the PCI bus in an efficient manner. It would also be very interesting to analyze the impact of peripheral device speed

on overall system performance. Rather than remove the peripheral device from the problem, it could be used to gather data on transfer size and FIFO size versus peripheral transfer rate and PCI Chip-set capabilities. This would be an involved and detailed investigation that should provide some interesting results.

Another interesting project would be to develop a device driver for this board for Windows NT™ or Windows 95™ Microsoft should be introducing their new Windows Driver Model or WDM in the very near future. Developing a driver for the PCI I/O Board using this new Model would be a worthwhile and rewarding effort.

5. Appendix

5.1 Appendix A - Glossary

5.2 Appendix B - Schematics

5.3 Appendix C - Programmable Logic Source

5.3.1 PCICTRL PLD

5.3.2 DMAC PLD

5.3.3 DECODE PLD

5.3.4 PARIO PLD

5.4 Bibliography

5.1.Appendix A - Glossary

ANSI - American National Standards Institute. A governing body which adopts and accepts standards in the United States.

ASIC - Application Specific Integrated Circuit.

bandwidth - the amount of data that can be delivered over a specified medium in a specified time.

burst - A data transfer mechanism that permits multiple contiguous pieces of data to be transferred without specifying the address information for each piece of data.

bus - A collection of related signals which are logically described as a single unit.

Chaining Mode A Mode of operation in which a DMA controller requires reduced CPU interaction. Once initiated the DMA controller can perform multiple DMA events using a linked-list or "chain" of control information.

CPLD - Complex Programmable Logic Device. A single chip which contains the logic of multiple common Programmable Logic Devices. These devices are typically very deterministic in propagation delay timings between input and output.

CPU Central Processing Unit. Typically used interchangeably with the term Microprocessor.

chipset - A group of standard chips which are typically used as a group to perform a function or set of functions. The core group of chips which are used in a PC-type system are referred to as a chipset.

DMA - Direct Memory Access. Direct access to Memory (and/or I/O) resources without CPU intervention. Typically a special device known as a DMA Controller is used to move large blocks of data between memory and/or I/O resources to reduce the workload of the microprocessor.

DRAM - Dynamic Random Access Memory. A relatively low cost volatile memory resource with a multiplexed address bus commonly used in

computers. This memory may be read from or written to in a random fashion.

EISA - Extended Industry Standard Architecture. An extended or enhanced version of the Industry Standard Architecture aimed as improving overall system performance.

FIFO Memory - First-In-First-Out Memory. This is a typically high speed memory resource which does not use unique addressing for it's data elements. This memory is accessed in a sequential, not random, fashion. The first item written to the memory is the first item read

FPGA - Field Programmable Gate Array. A device similar to a CPLD that is typically more flexible, but also less deterministic in it's timing results.

GUI - Graphical User Interface. A user interface typically operated with a touch-screen or some sort of pointing device such as a mouse or pen. This contrasts with a console interface which requires the operator to enter commands with a keypad or keyboard.

IEEE Institute of Electrical and Electronics Engineers.

Initiator - In the PCI specification this term is used interchangeably with Master. An initiator is a device which gains control of the bus and begins or initiates a bus cycle.

I/O - Input/Output.

ISA - Industry Standard Architecture. The architecture developed by International Business Machines (IBM) and used in the first Personal Computers.

master - A device which resides on a bus and may gain control of the bus.

Megabyte - 2^{20} or 1,048,576 bytes

Multibus - A bus specification developed by Intel Corporation in the 1970's commonly used for industrial applications.

Multibus II A bus specification developed by Intel Corporation in the early 1980's as a 32-bit more robust upgrade to Multibus.

OEM - Original Equipment Manufacturer. A company that manufactures any type of equipment under their own brand name and sells to an end customer in the retail marketplace.

PCI - Peripheral Component Interconnect. A high speed bus specification developed by a consortium of companies, spearheaded by Intel Corporation in an effort to increase the performance of the Personal Computer.

platform - The architectural basis of a computer system. For example an ISA platform is a computer based on the ISA specification.

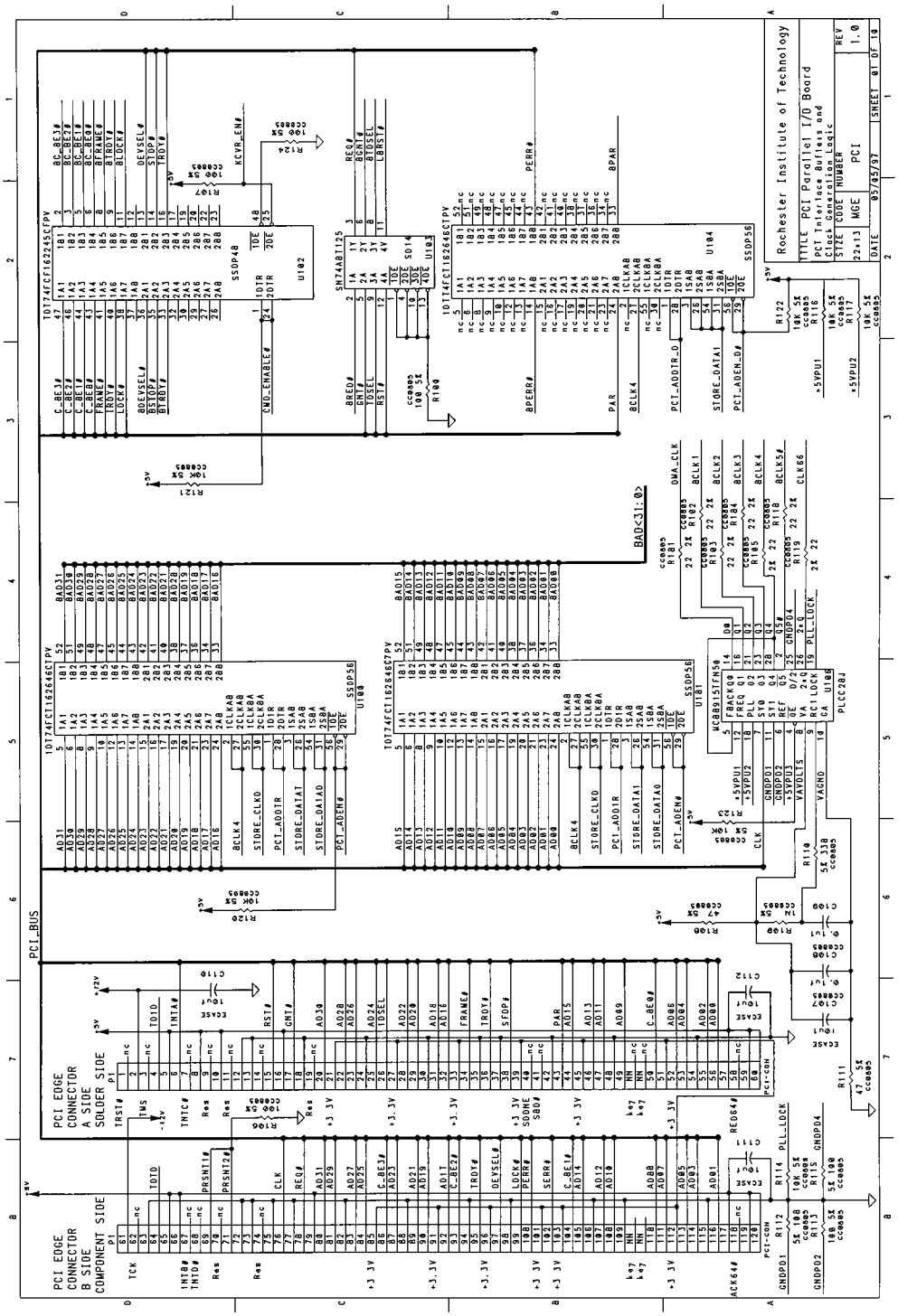
S-Bus - A high speed synchronous bus developed by Sun Microsystems Inc.

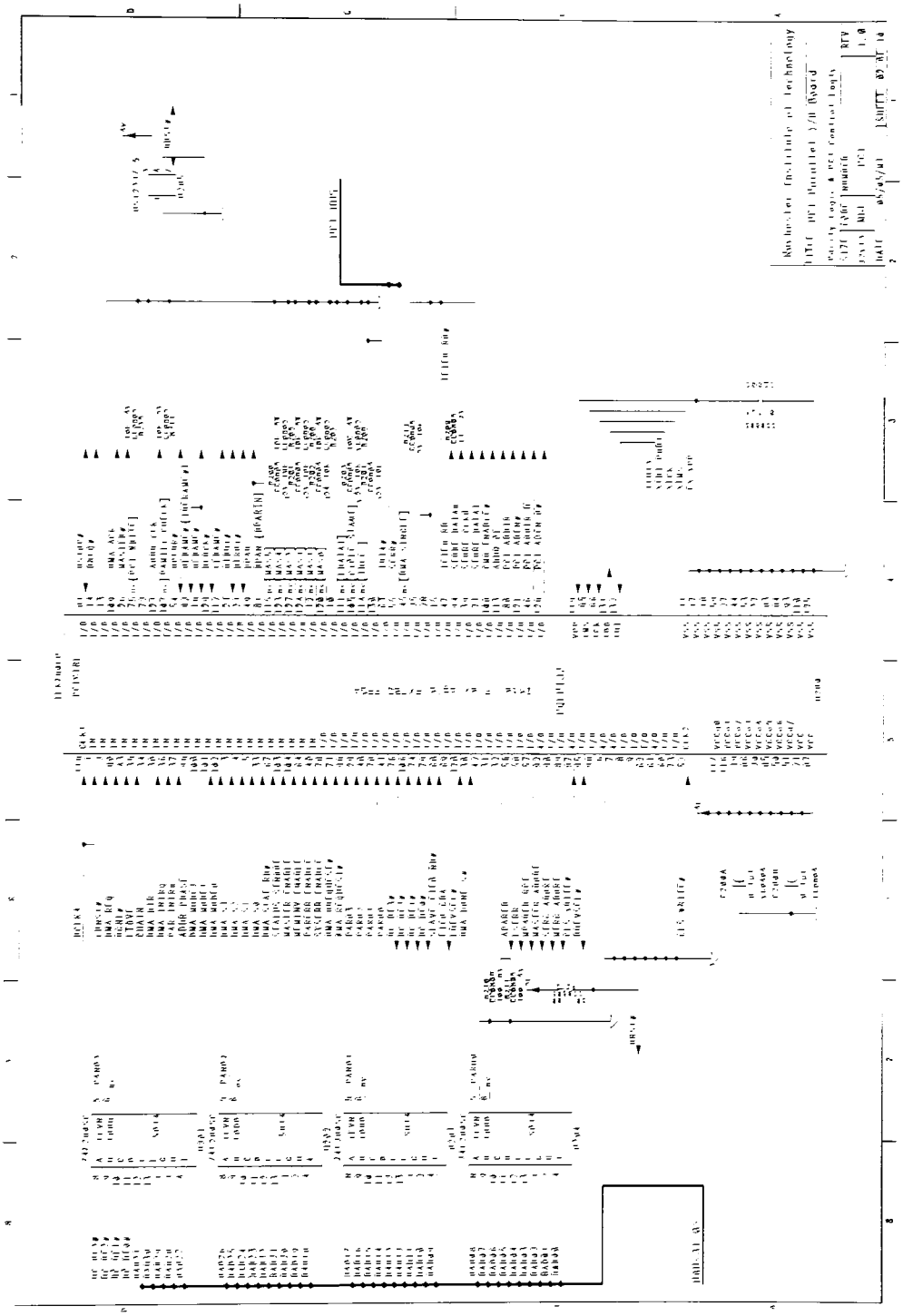
Scatter-Gather - A technique often used in imaging applications which causes different data sets of potentially different sizes to be grouped separately both logically, and physically in memory. The different data sets are stored in a "scattered" arrangement, but then "gathered" together to render the final image. A DMA Controller must support chaining in order to effectively implement this feature.

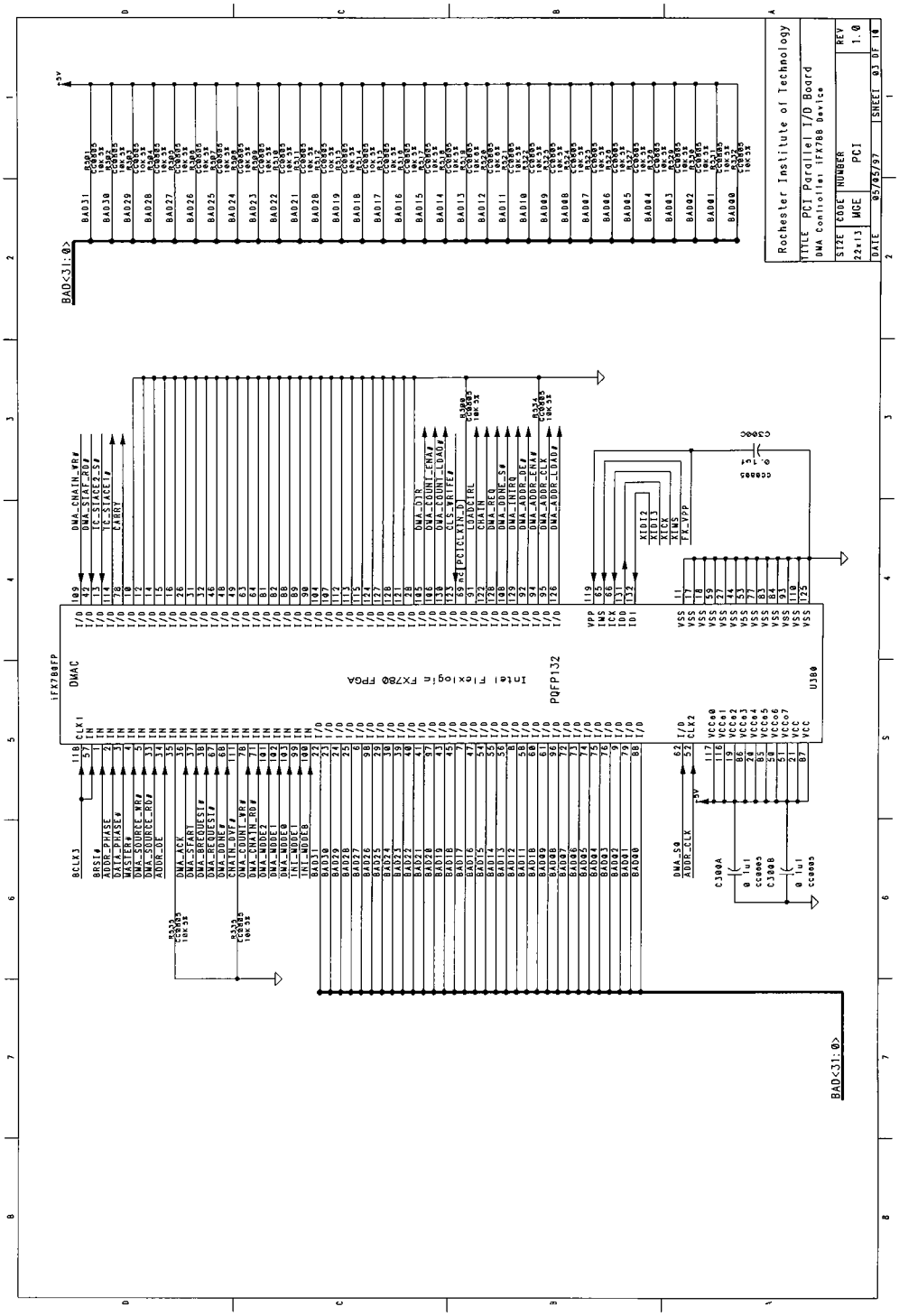
Target - In a PCI system the target is the device being addressed by the Initiator or Master. Synonymous with Slave in a typical master-slave system.

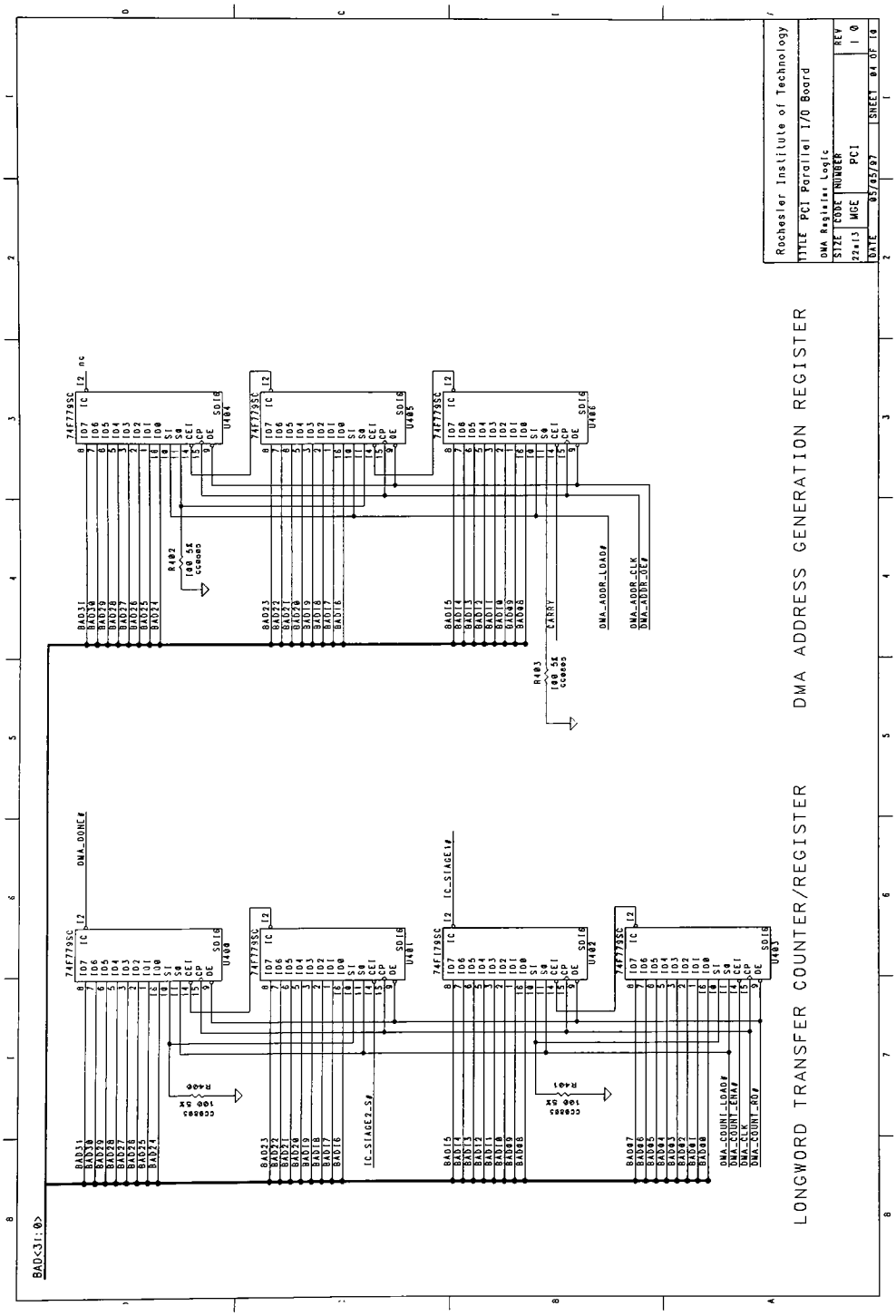
VMEbus - A 32-bit bus standard spearheaded by Motorola in the late 70's early 80's meant to dethrone Multibus as the popular bus standard.

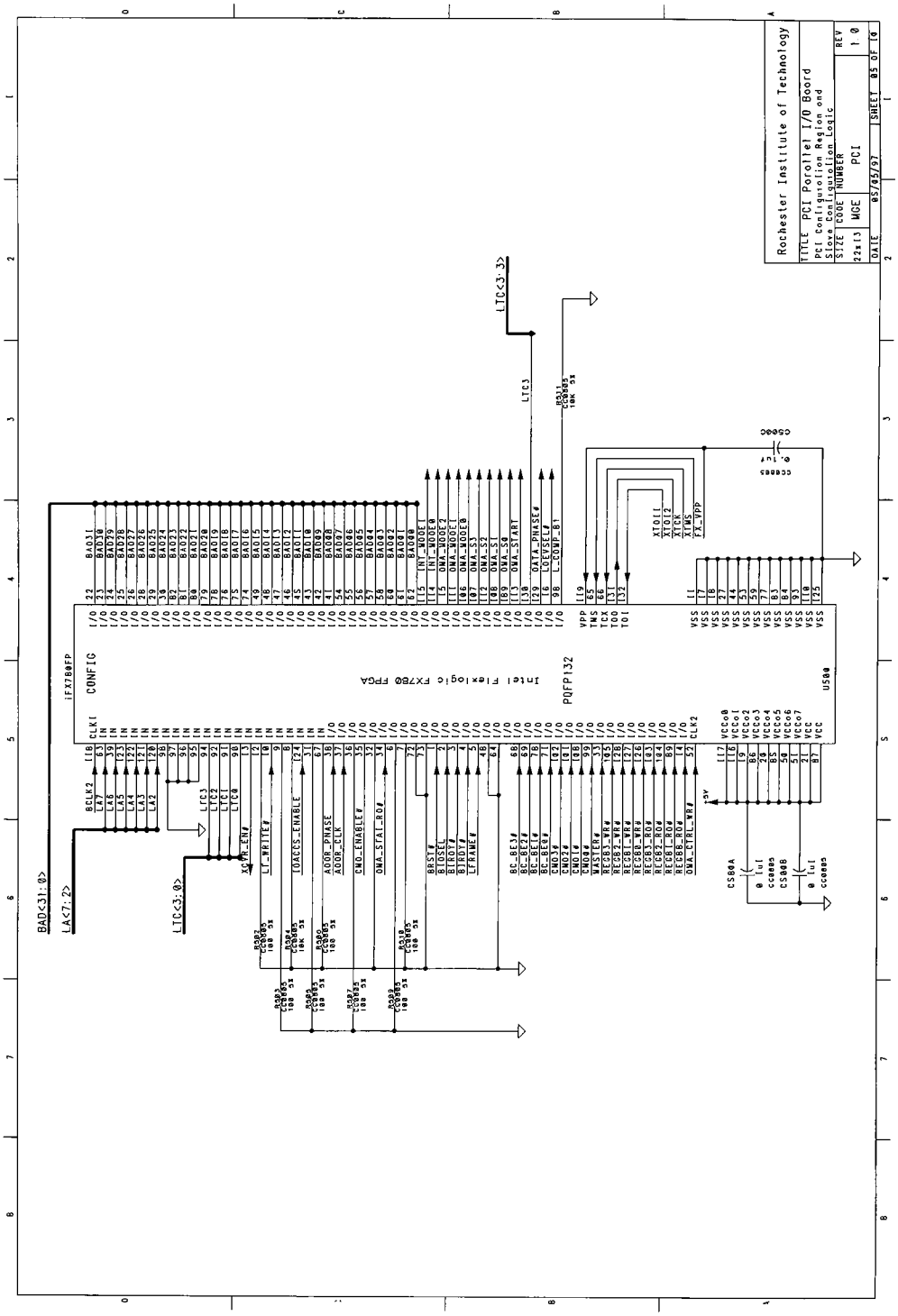
5.2.Appendix B - Schematics

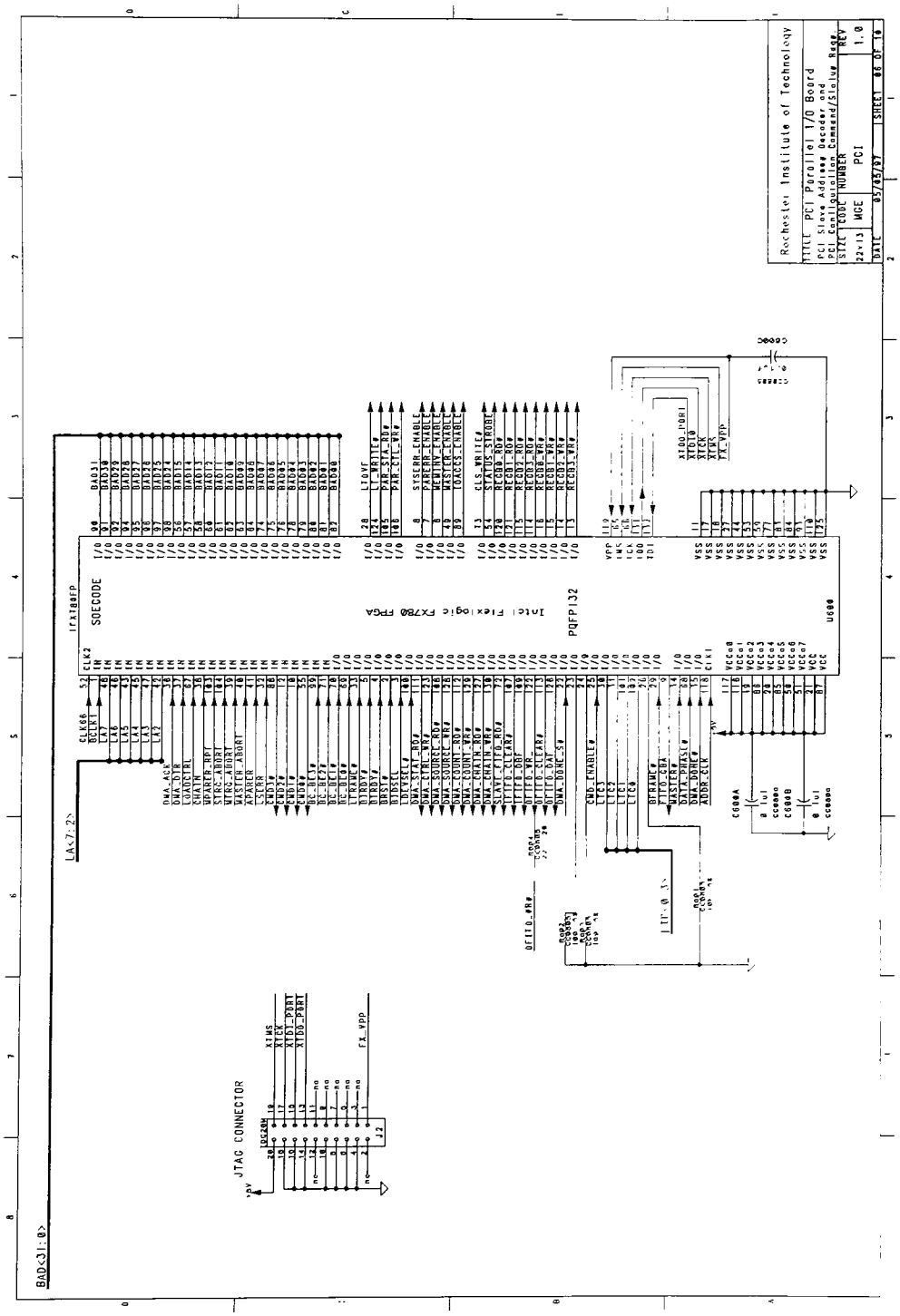


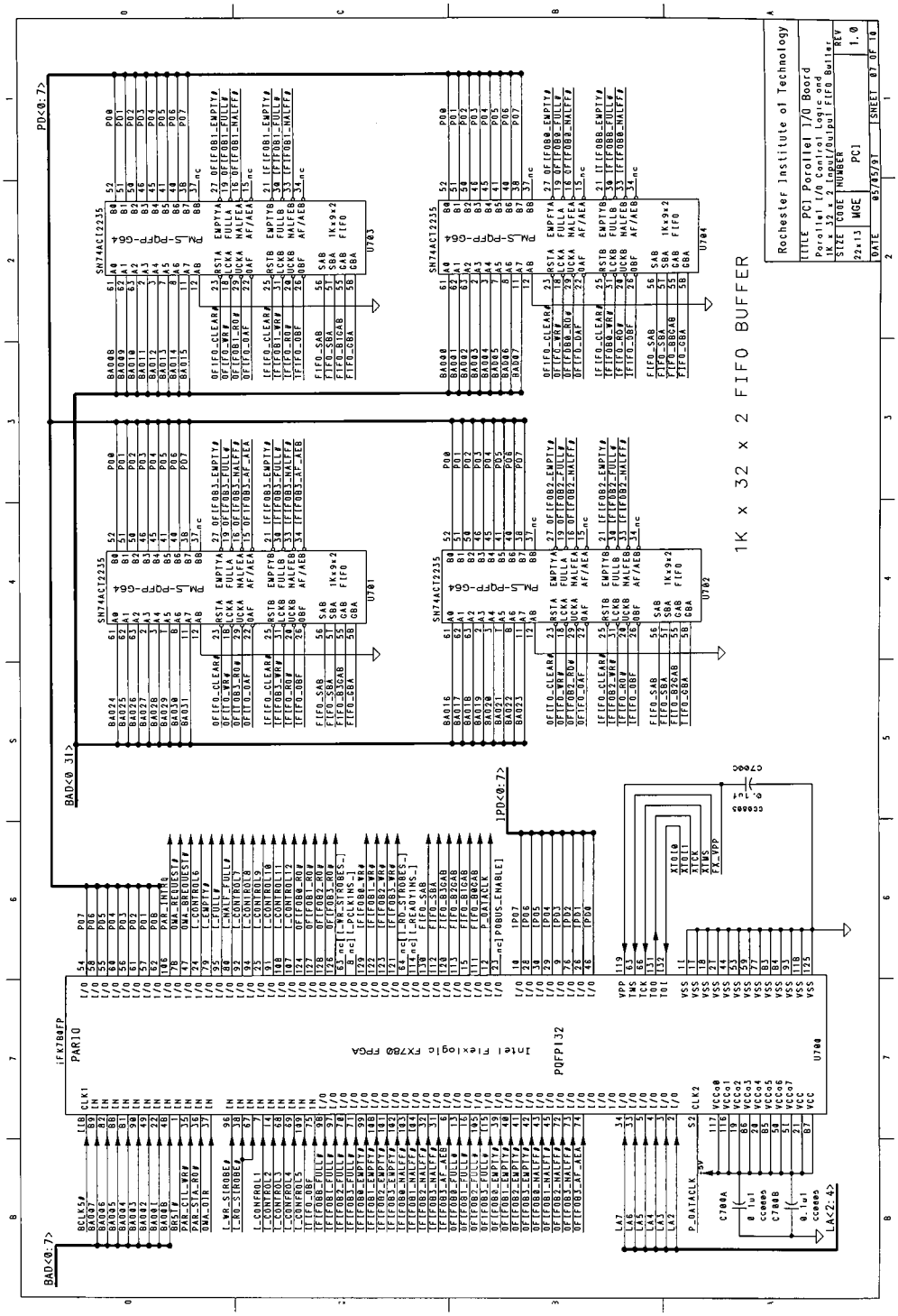



Rochester Institute of Technology
TITLE PCI Parallel I/O Board
DWA Controller iFX780 Device
SIZE 22x13 | CODE NUMBER WGE | PCI | REV 1.0
DATE 05/05/97 | SHEET 03 OF 10
iFX780FP
DMAC
Intel Flexlogic FX780 FPGA
PQFP132
U300
BAD<31:0>
BCLK3
DNA_CHAIN_WR#
DNA_START_RD#
TC_START#
TC_STATE#
CARRY
DNA_ACK
DNA_SFART
DNA_BREQUEST#
DNA_REQUEST#
CHAIN_DVF#
DNA_COUNT_WR#
DNA_CHAIN_RD#
DNA_MODE2
DNA_MODE1
DNA_MODE0
INI_MODE1
INI_MODE0
DNA_COUNT_ENA#
DNA_COUNT_LOAD#
TC_STATE#
LOADCTRL
CHAIN
DNA_REQ
DNA_DRIVER_S#
DNA_ADDR_DF#
DNA_ADDR_ENA#
DNA_ADDR_CLK
DNA_ADDR_LOAD#
XIDI2
XIDI3
XIN5
FX_VPP
DNA_S0
ADDR_CLK
C3900
C3908
C3906


DMA ADDRESS GENERATION REGISTER
LONGWORD TRANSFER COUNTER/REGISTER
Rochester Institute of Technology
TITLE PCI Parallel I/O Board
DMA Register Logic
SIZE CODE NUMBER REV
22x13 WGE PCI 1 0
DATE 05/05/97 SHEET 04 OF 10
BAD<31:0>
74F779SC
DMA_DONE#
IC_STAGE2_S#
IC_STAGE1#
DMA_ADDR_LOAD#
DMA_ADDR_CLK
DMA_ADDR_OE#
CARRY
DMA_COUNT_LOAD#
DMA_COUNT_ENA#
DMA_CLK
DMA_COUNT_RQ#


Rochester Institute of Technology
TITLE PCI Parallel I/O Board
PCI Configuration Region and
Slave Configuration Logic
SIZE CODE NUMBER REV
22x13 WOE PCI 1 0
DATE 05/05/97 SHEET 05 OF 10
Intel Flex10K FX780 FPGA
POFP132
U500
CONFIG
BAD<31:0>
LA<7:2>
LTC<3:0>
LTC<3:3>


Rochester Institute of Technology
TITLE PCI Parallel I/O Board
PCI Slave Address Decoder and
PCI Configuration Command/Status Regs
SIZE CODE NUMBER REV
22x13 MGE PCI 1.0
DATE: 05/05/97 SHEET 06 OF 10
Intel FlexLogic FX780 FPGA
PQFP132
SOECODE
JTAG CONNECTOR
BAD<31:0>
LA<7:2>
J2


1K x 32 x 2 FIFO BUFFER
Rochester Institute of Technology
TITLE PCI Parallel I/O Board
Parallel I/O Control Logic and
1K x 32 x 2 Input/Output FIFO Buffer
SIZE CODE NUMBER REV
22x13 MGE PCI 1.0
DATE 05/05/97 SHEET 07 OF 10
SN74ACT2235
PM_S-PQFP-64
U701
U702
U703
U704
Intel Flexlogic FX780 FPGA
PQFP132
U700
IFX780FP
PAR10

5.3. Appendix C - Programmable Logic Source Code

5.3.1. PCICTRL PLD

```
module PCICTRL flag '-r3'
title 'PCI Interface Master/Slave Controller
Sheet 1 of 1, Schematic Diagrams
"-------------------------------------------------------------------------------
"  Description:
"
"  This FPGA implements the following functionality on the PCI
"  I/O Board:
"
"    Generation of the PCI Bus Control and error Signals when in Master Mode.
"    This is performed by the PCI Master State Machine.
"
"    Generation of Slave Mode PCI Bus signals.
"
"    Control of Arbitration PCI Master Mode Transfers.
"
"    Control of PCI Bus Transceivers for Master and Slave accesses.
"
"    Provides PCI Status Information to Configuration Region.
"
"    Provides Parity Error Detection and Notification based upon Parity
"    Register input values.
"
"  Output and Bi-directional Signal Descriptions:
"
"   STATUS_STROBE_SYN(O):  Synchronized PCI Status Strobe.  This signal is a
"   synchronized version of the STATUS_STROBE signal which is generated by
"   the SDECODE PLD during a valid PCI Configuration Space Read decoded at
"   the Status Register Address.  This registered output is synchronized to
"   the Buffered PCI Bus Clock BCLK.
"
"   MPARER_RPT(O):  Master Parity Error Reporting.  This signal is driven to an
"   active high state when the PCI I/O Interface Board is a PCI Master and
"   the PCI Parity signal PERR# is asserted.  This output is connected to the
"   SDECODE PLD and remains active until a Status Register Read is performed
"   to the PCI Configuration Region on this board by another PCI Master.
"   This registered output is synchronized to the Buffered PCI Bus Clock
"   BCLK.
"
"   MTRG_ABORT(O):  PCI Target Abort Received.  This signal is asserted when the
"   Interface Board is a PCI Master and a Target Abort is issued by
"   the PCI Target being accessed.  This output is connected to the SDECODE
"   PLD and remains active until a Status Register Read is performed to the
"   PCI Configuration Region on this board by another PCI Master.  This
"   registered output is synchronized to the Buffered PCI Bus Clock BCLK.
"
"   STRG_ABORT(O):  PCI Target Abort Issued.  This signal is asserted when the
"   Interface Board is a PCI Target, and this board issues a Target
"   Abort.  This output is connected to the SDECODE PLD and remains active
"   until a Status Register Read is performed to the PCI Configuration
"   Region on this board by another PCI Master.  This registered output is
"   synchronized to the Buffered PCI Bus Clock BCLK.
"
"   MASTER_ABORT(O):  PCI Master Abort Issued.  This signal is generated when
"   the Parellel Interface Board is a PCI Master, and the addressed Target
"   does not respond by asserting DEVSEL# by Clock #6 of the PCI Cycle.  This
"   output is connected to the SDECODE PLD and remains active until a Status
"   Register Read is performed to the PCI Configuration Region on this board
"   by another PCI Master.  This registered output is synchronized to the
"   Buffered PCI Bus Clock BCLK.
"
"   MAS5,MAS4, MAS3,MAS2,MAS1,MAS0(O): PCI Control State Machine State
"   Control Pins.  This State Machine Controls PCI Interfacing on the Parallel
"   Interface Board.  Refer to the PCI Controller State Machine State Diagram in
"   this Document.  This State Machine is clocked by the Buffered PCI Bus Clock
"   BCLK.
"
"   LDATA1,IDLE_(O):  Last Data Phase Detection State Machine Control Pins.
"   This State Machine Performs Detection of the final Data Phase in a Burst
"   PCI Transaction.  This Data Phase takes place after the PCI FRAME#
"   signal has been deasserted.  The signal LDATA1 is only active during
"   the last DATA Phase of a given PCI Bus Transaction (i.e. FRAME# is High
"   and IRDY# and TRDY# have not yet been sampled active).  The signal IDLE_
"   is Low when the PCI Bus is Idle.  Refer to the PCI Last Data Phase
"   Detection State Machine in this document.  This State Machine is clocked
"   by the Buffered PCI Bus Clock BCLK.
"
"   BFRAME_, IBFRAME_(I/O):  Buffered PCI FRAME# Output Pin.  This signal is
"   asserted to initiate a Master (initiator) cycle on the PCI Bus by the
```

```
"   Interface Board.  It is an input when the board is not a Master, and is driven as
"   an output when the board is a Master.  IBFRAME_ is used to enable both
"   pin and register feedback into the global macrocell.  The same pin is indicated
"   for both signal names.
"
" - BIRDY_(I/O):  Buffered PCI Initiator Ready#.  This signal is asserted to indicate
"   that the PCI bus transaction Initiator is Ready for either a Read or Write Cycle.
"   This signal is driven as an output when the Interface Board is a PCI
"   and is an input when a PCI Slave.
"
"   BTRDY_(I/O):  Buffered PCI Target Ready#.  This signal is asserted to indicate
"   that the PCI bus transaction Target is Ready during either a Read or Write
"   Cycle.  This signal is driven as an output when the Interface Board is
"   a PCI Target, and is an input when a PCI Master.
"
"   BSTOP_(I/O):  Buffered PCI Stop#.  This signal is asserted by a PCI Target to indicate
"   a request for the initiator to stop the current transaction.  This pin is driven
"   high as
"   an output when the Interface Board is a slave (a stop is never requested by
"   the Interface Board).  When received as a Master, the current transaction
"   is stopped by the Interface Board.
"
"   BPAR,BPARIN(I/O):  Buffered PCI Parity.  This signal indicates even parity across AD[31..0]
"   and C/BE[3..0]# for address and data phases of all PCI Transactions.  This signal is
"   driven valid one clock after the address phase by the Master, and one clock after IRDY#
"   is asserted on a write transaction (by Initiator) or one clock after TRDY# is asserted
"   on a read transaction (by Target).  This signal is used in conjunction with the BAD and BC/BE
"   signals to determine if a Parity Error has occurred.  BPARIN is used to enable both registered
"   and pin feedback.
"
"   BREQ_(I):  Buffered PCI Request#.  This signal indicates to the PCI central resources arbiter
"   that the Interface Board desires to use the bus.  This is an Output only on the
"   Interface Board.
"
"   BC_BE3_.. BC_BE0_(I/O):  Buffered PCI Command/Byte Enable [3..0]#.  These multiplexed
"   signals are driven by the PCI Initiator.  During the address phase they contain cycle type
"   information, during the data phase they are used as byte enables.  The Interface board
"   always drives these signals active low during the data phase, as all transactions are 32-bits.
"   During the address phase they are driven based upon the DMA Cycle definition bits as
"   programmed in the DMA Control Register.
"
"   BDEVSEL_(I/O):  Buffered PCI Device Select#.  This signal is driven active by the
"   Interface Board when it is accessed by a PCI Master to indicate that it will be responding to
"   the access.  When a Master, this signal is received to indicate that a slave is responding to the
"   access initiated by the Interface Board.
"
"   LDEVSEL_(O):  Local Device Select#:  This signal is generated on board to indicate to the
"   on-board logic that the current PCI Bus cycle generated by another is meant for the
"   Interface Board.
"
"   BAD[7..0] (I):  Buffered Address/Data Bit 7. The Buffered Address/Data bus bits are inputs
"   to this device, used to load the Cache Line Size into the Cache Burst Counter.  This counter
"   is used on the Interface Board for Memory Write and Invalidate and Memory Read
"   Line PCI Transaction Types.
"
"   CLS_WRITE_(I):  CLEAR STROBE WRITE#.  This signal is used to Clear the
"   Interrupt Signal (INTA#) when it has been latched onto the PCI Bus.
"
"   CBC[7..0] (N):  Cache Burst Counter 7..0:  These internal nodes are used to store the Cache
"   Line Size Count for Memory Write and Invalidate and Memory Read Line PCI Transaction
"   Types.  The stored value is used as a starting point for the down count.  When the down count
"   reaches zero, the PCI transaction will be terminated.
"
"   BC3U..BC0U & BC3L..BC0L(N):  8-bit Cache Burst Counter.  This 8-bit counter is implemented
"   as two 4-bit counters with underflow.  When the low order counter (BC3L..BC0L) reaches zero
"   the higher order counter will decrement.  When the entire 8-bit count expires, the PCI
"   transaction will terminate.
"
"   LOWUFLOW(O):  Low Count Underflow.  This signal is driven active high when the low order
"   nibble of the Cache Burst Counter reaches zero.
"
"   BURST_DONE(O):  Burst Done.  This signal is driven active high when the 8-bit Cache Burst
"   Counter expires indicating that the current PCI transaction should be completed with the current
"   data phase.
"
"   CMD_ENABLE_(O):  Command Enable.  This signal is used to control the direction of the PCI Bus
"   Command Signal Transceiver.  The PCI Command Signal Set is always driven by the current PCI
"   Transaction Initiator, so this signal is used to envelope the Initiator Transaction cycle when the
"   PCI I/O board is a Master.
"
"   ADDR_PHASE(O):  Address Phase.  This signal is driven active high for one BCLK Period during
"   the PCI Bus address phase when the PCI I/O board is a Master.
"
"   PCI_WRITE(O):  PCI Write Detect.  This signal is driven active high to indicate that the
```

```
"   current PCI transaction is a PCI Write.  When the PCI I/O board is a Master,
"   this signal is driven at the start of the Address Phase and held until the bus is
"   relinquished.  When the PCI I/O board is a slave, this signal is driven one
"   BCLK Period following the Address Phase, and deasserted once the PCI Bus
"   returns to Idle.
"
"   DMA_ACK(O):  DMA Cycle Acknowledge.  This signal is driven active high to indciate
"   to the DMA Control Logic that a succesful Data transfer has taken place on the PCI Bus.
"   A successful DMA transfer is indicated by IRDY# and TRDY# being asserted at the same
"   time.
"
"   MASTER_(O):  Master#.  This signal is driven active low to indicate that the Parallel
"   Interface Board is currently the PCI Bus Master and that a PCI transaction is in Progress.
"   Master# is asserted at the beginning of the Address Phase and held active until the end of
"   the PCI transaction.
"
"   DMA_MEM_RD_(I):  Memory Read Command.  This signal indicates that the On-board DMA
"   Controller is generating a Memory-to-Memory transaction across the PCI Bus, and that the
"   direction is from System Memory to the PCI I/O board.
"
"   DMA_MEM_WR_(I):  Memory Write Command.  This signal indicates that the On-Board
"   DMA Controller is generating a Memory-to-Memory transaction across the PCI Bus, and that
"   the direction is from the PCI I/O board to System Memory.
"
"   INTA_(OD):  PCI Interrupt A#:  This open drain output is used to generate an interrupt across
"   the PCI Bus.  The two interrupt sources are the DMA Contoller (end of chain, or terminal count)
"   or the PCI Parallel Interface Interrupt Request.
"
"   PCI_ADDIR(O):  PCI Address/Data Direction Control.  This signal controls the direction of the
"   PCI Address/Data transceivers.  When this signal is driven high the PCI I/O board is
"   driving the PCI Address/Data Bus.  When this signal is driven low the PCI Bus is being driven onto
"   the PCI I/O board BAD Bus.
"
"   PCI_ADEN_(O):  PCI Address/Data Bus Enable.  This signal controls the output enable pins of the
"   PCI Address/Data Transceivers.  When this signal is driven low the transceivers are in a high-
"   impedance state.  This signal is part of the PCI Bus control State Machine.
"
"   ADDR_OE (O):  Address Output Enable.  This signal Controls the Address Output Enable of the on-
"   board DMA Controller.  During the
"-------------------------------------------------------------------------------------------------
"                           Declarations
"-------------------------------------------------------------------------------------------------

                    PCICTRL    device 'ifx780fp' ; " Intel FX780 PQFQ Device

"-------------------------------------------------------------------------------------------------
"                           Input Pins
"-------------------------------------------------------------------------------------------------

        BCLK                    pin 118 ;  "33Mhz PCI Bus Clock
        PCI_CLKIN               pin   1 ;  "33Mhz PCI Bus Clock for Equation use
        LBRST_                   pin   2 ;  "Buffered Active Low PCI Reset
        DMA_S3                  pin   3 ;  "DMA Cycle Definition Bit 3
        DMA_S2                  pin   4 ;  "DMA Cycle Definition Bit 2
        DMA_S1                  pin   5 ;  "DMA Cycle Definition Bit 1
        DMA_S0                  pin  33 ,  "DMA Cycle Definition Bit 0
        CHAIN                   pin  34 ,  "DMA Controller Chaining Status
        LTOVF                   pin  35 ,  "Latency Timer Overflow Input
        DMA_INTRQ               pin  36 ,  "DMA Controller Interrupt Request
        PAR_INTRQ               pin  37 ,  "Parallel Interface Interrupt Request

        BPERR_                  pin  54 ,  "Buffered PCI Parity Error#
        IBPERR_                 pin  54 ,  "Buffered PCI Parity Error Input#
        DMA_REQ                 pin  80 ,  "Local DMA Start Request
        DMA_SINGLE              pin  45 ,  "PCI Single Data Phase Request
        DMA_DONE_               pin  30 ,  "Local DMA Transfer Completed#
        BGNT_                   pin  43 ,  "Buffered PCI Bus Grant#
        DMA_DIR                 pin  38 ,  "DMA Cycle Write/Read#

        DMA_STAT_RD_            pin  67 ,  "DMA Control/Status Register Read#

        SLAVE_FIFO_RD_          pin  68 ,  "FIFO Read Cycle from Another Master#
        FIFO_GBA                pin  69 ,  "FIFO Read Enable Input

        SYSERR_ENABLE           pin  70 , " SERR_ Generation Enable


"-------------------------------------------------------------------------------------------------
"                     DMA Request & Mode Register Inputs
"-------------------------------------------------------------------------------------------------
        DMA_BREQUEST_           pin  71 ;  "DMA Burst Request Input#
        DMA_REQUEST_            pin  99 ;  "DMA Non-Burst Request Input#
        DMA_MODE2               pin 100 ;  "DMA Mode Register Bit 2
        DMA_MODE1               pin 101 ,  "DMA Mode Register Bit 1
```

```
        DMA_MODE0               pin 102 ;  "DMA Mode Register Bit 0
"-------------------------------------------------------------------------------
"                       PCI Valid Command Register Enable Inputs from SDECODE
"-------------------------------------------------------------------------------
        STATUS_STROBE           pin 103 ; "Status Read Strobe from Slave Decoder
        STATUS_STROBE_SYN      node  42 ; "Status Read Strobe from Slave Decoder Synch'd to BCLK
        MASTER_ENABLE           pin 104 ; "PCI Command Register Master Enable
        MEMINV_ENABLE           pin  64 ; "PCI Command Register Memory Write and Invalidate En.
        PARERR_ENABLE           pin  40 ; "PCI Command Register Parity Error Enable
"-------------------------------------------------------------------------------
"                       PCI Error Reporting Logic
"-------------------------------------------------------------------------------
        MPARER_RPT              pin  57 , "PCI Bus Master Parity Error Report
        MTRG_ABORT              pin  89 , "PCI Target Abort Received as Master
        STRG_ABORT              pin  90 ; "PCI Target Abort Issued as Target
        MASTER_ABORT            pin  92 , "Master Abort Received as Target
        APARER                  pin  56 , "Any Parity Error Detected

        SERR_                   pin  55 , "PCI System Error# (Open Drain)
        ISERR                   pin  58 , "PCI System Error Issued as Master

"-------------------------------------------------------------------------------
"                               Bi-Directional Pins
"-------------------------------------------------------------------------------
        MAS5                    pin 115 ,  "DMA Master State Machine S5
        MAS4                    pin 123 ;  "DMA Master State Machine S4
        MAS3                    pin 122 ;  "DMA Master State Machine S3
        MAS2                    pin 124 ;  "DMA Master State Machine S2
        MAS1                    pin  12 ;  "DMA Master State Machine S1
        MAS0                    pin 120 ;  "DMA Master State Machine S0

        LDATA1                  pin 111 ;  "Last Data Detection State Machine S1
        IDLE_                   pin 114 ;  "Last Data Detection State Machine S0

        .BFRAME_                 pin  16 , "Buffered PCI FRAME# Output pin
        IBFRAME_                pin  82 , "Buffered PCI FRAME# Input pin
        BIRDY_                  pin  24 ; "Buffered PCI Initiator Ready#
        IBIRDY_                 pin  24 ; "Buffered PCI Initiator Ready#
        BTRDY_                  pin  22 ; "Buffered PCI Target Ready#
        BTRDY_D                 node 30 ; "Buffered PCI Target Ready Delayed#
        BTRDY_DD                node 31 ; "Buffered PCI Target Ready Delayed Again#
        BTRDY_DDD               node 32 ; "Buffered PCI Target Ready Delayed Again , Again#
        IBTRDY_                 pin  22 ; "Buffered PCI Target Ready#
        IBSTOP_                 pin  91 ; "Buffered PCI STOP# Input
        BSTOP_                  pin  91 ; "Buffered PCI STOP#
        BPAR_                   pin  49 , "Buffered PCI PARITY#
        BPARIN_                 pin  81 , "Buffered PCI PARITY#
        BREQ_                   pin  14 , "Buffered PCI Bus Request#
        BC_BE0_                 pin  79 , "Buffered PCI Byte Enable 0#
        BC_BE1_                 pin  74 ; "Buffered PCI Byte Enable 1#
        BC_BE2_                 pin 106 ; "Buffered PCI Byte Enable 2#
        BC_BE3_                 pin  76 ; "Buffered PCI Byte Enable 3#
        BDEVSEL_                pin  95 ; "Buffered PCI Device Select#
        BDEVSELI_               pin  95 ; "Buffered PCI Device Select Input Only#
        LDEVSEL_                pin 128 ; "Local DEVICE  Selected by PCI Master#
        CLS_WRITE_              pin  97; "Clear (INTA) Strobe Write#

"-------------------------------------------------------------------------------
"                               Outputs
"-------------------------------------------------------------------------------
        CMD_ENABLE_             pin 108 , "Control for Command Transceiver#
        ADDR_PHASE              pin  96 , "PCI Address Phase Detected
        ADDR_CLK                pin 127 , "PCI Address/CMD Latch Control
        PCI_WRITE               pin  75 , "PCI Write Cycle Detect#
        DMA_ACK                 pin 109 , "DMA Transfer Acknowledged
        MASTER_                 pin  26 ; "PCI Master Status#
        LFRAME_                 pin 112 ; "Local FRAME# (Envelopes PCI Cycle)
        INTA_                   pin  63 ; "PCI Interrupt Pin A# (Open Drain)

        PCI_ADDIR               pin  88 ; "PCI Address/Data Direction Control
        PCI_ADDIR_D             pin  46 ; "PCI Address/Data Direction Control Delayed
        PCI_ADEN_               pin 121 ; "PCI Address/Data Bus Output Enable#
        PCI_ADEN_D              pin 126 ; "PCI Address/Data Bus Output Enable Delayed#
        STORE_DATAI             pin  23 ; "PCI Address/Data Transceiver Register Input Storage Ctrl.
        STORE_DATAO             pin  94 ; "PCI Address/Data Transceiver Register Output Storage Ctrl.
        STORE_CLKO              pin  39 ; "PCI Data Output Storage Clock Strobe

        ADDR_OE                 pin 113 ; "Address Output Enable for DMA Controller

        FIFO_RD_ENA            node  29 ; "On-board FIFO Read Enable Control
        IFIFO_RD_               pin  47 ; "On-board FIFO Read Strobe#

        DMA_DONE_S_             pin 30; "DMA DONE Internally Sync'd
```

```
        CYCLE_START             pin 105;" Master Cycle State Machine Start

        BRST_                  pin 98; "Local Open Collector Buffered PCI RESET

"-------------------------------------------------------------------------------------------
"       Pins Associated With Parity Generation and Error Detection
"-------------------------------------------------------------------------------------------
        IPAR_                  node  48 , "Internal Parity Generation
        PARO_3                  pin  29 ; "Parity Logic Output 3
        PARO_2                  pin  48 , "Parity Logic Output 2
        PARO_1                  pin  78 , "Parity Logic Output 1
        PARO_0                  pin  41 , "Parity Logic Output 0
        PARITY_CHECK            pin 107 , "Parity Checking Clock Cycle

"-------------------------------------------------------------------------------------------
"                 Set Definitions and Constants
"-------------------------------------------------------------------------------------------


        H,L,X,C,Z,K   1,0,.X.,.C.,.Z.,.K.;
        SIMULATE      0;

        PCI_ADDIR_D istype  'buffer,reg,pos';
        PCI_ADDIR   istype  'com';
        STORE_DATAI istype  'buffer,reg,pos';
        CMD_ENABLE_ istype 'reg':

        BREQ_       istype  'reg':
        BFRAME_     istype  'invert,reg,neg';
        LFRAME_     istype  'reg':
        MASTER_     istype  'reg';
        ADDR_OE     istype  'buffer,reg,pos';
        PCI_ADEN_   istype  'reg';
        PCI_ADEN_D  istype  'reg';
        MAS5        istype  'buffer,reg,pos';
        MAS4        istype  'buffer,reg,pos';
        MAS3        istype  'buffer,reg,pos';
        MAS2        istype  'buffer,reg,pos';
        MAS1        istype  'buffer,reg,pos';
        MAS0        istype  'buffer,reg,pos';

        LDATA1     istype 'buffer,reg,pos';
        IDLE_      istype 'buffer,reg,pos';

        FIFO_RD_ENA istype 'buffer,reg,pos';

        DMA_Mode                  [DMA_MODE2,DMA_MODE1,DMA_MODE0];
        FlyBy_Up_Normal           [    0    ,    0    ,    0    ];
        FlyBy_Dwn_Normal          [    0    ,    0    ,    1    ];
        FlyBy_Up_Chain            [    0    ,    1    ,    0    ];
        FlyBy_Dwn_Chain           [    0    ,    1    ,    1    ];
        MemMem_Up_Normal          [    1    ,    0    ,    0    ];      "Not Supported
        MemMem_Dwn_Normal       = [    1    ,    0    ,    1    ];      "Not Supported
        Invalid1                = [    1    ,    1    ,    0    ];      "Not Valid
        Invalid2                = [    1    ,    1    ,    1    ];      "Not Valid
"-------------------------------------------------------------------------------------------
"       Intel Property Statements for Open Drain Outputs
"-------------------------------------------------------------------------------------------
        Intel PROPERTY '@PIN_ATTRIB INTA_ OPEN_DRAIN';
        Intel PROPERTY '@PIN_ATTRIB SERR_ OPEN_DRAIN';
        Intel PROPERTY '@PIN_ATTRIB BRST_ OPEN_DRAIN';

"-------------------------------------------------------------------------------------------
"       Definition of Burst Counter Register
"-------------------------------------------------------------------------------------------
"        CurBurstData    [BAD3,BAD2,BAD1,BAD0];
"        CurCount        [BC3L,BC2L,BC1L,BC0L];

"-------------------------------------------------------------------------------------------
"                 PCI Transfer Command Definitions
"-------------------------------------------------------------------------------------------
        IO_Read         !BC_BE3_ & !BC_BE2_ &  BC_BE1_ & !BC_BE0_;
        IO_Write        !BC_BE3_ & !BC_BE2_ &  BC_BE1_ &  BC_BE0_;

        PCI_Cmd         [BC_BE3_,BC_BE2_,BC_BE1_,BC_BE0_];
        DMACycle        [DMA_S3,DMA_S2,DMA_S1,DMA_S0];
        MemRD           [  0  ,   1   ,   1   ,   0   ];
        MemRDMult       [  1  ,   1   ,   0   ,   0   ];
        MemRDLine       [  1  ,   1   ,   1   ,   0   ];
        IO_RD           [  0  ,   0   ,   1   ,   0   ];
        IO_WR           [  0  ,   0   ,   1   ,   1   ];
        Config_Read     [  1  ,   0   ,   1   ,   0   ];
        Config_Write    [  1  ,   0   ,   1   ,   1   ];
```

```
        MemWR           [  0  ,   1   ,  1   ,  1   ];
        MemWRInv        [  1  ,   1   ,  1   ,  1   ];
        Off             [  Z  ,   Z   ,  Z   ,  Z   ];
        On              [  0  ,   0   ,  0   ,  0   ];

"-------------------------------------------------------------------------------------------------
"                State Machine State Definitions
"-------------------------------------------------------------------------------------------------
        PCI_Control          [BFRAME_,ADDR_OE,PCI_ADEN_,MAS5,MAS4,MAS3, MAS2, MAS1, MAS0];
        Idle                 'b100000000;
"Master Access Cycles
        Drive_Address      = ^b110000000; "
        Drive_FRAME        = ^b010000001; "
        Read_Turn            ^b000000001; "
        MDATA                'b000000011; "
        Last_Burst_Xfer    = ^b100000010; "
        WaitDevice0        = ^b000001010; "
        WaitDevice1        = ^b000011010; "
        WaitDevice2        = ^b000011110; "
        WaitDevice3          ^b000011110; "
        MABORT               'b100001000; "
"Slave Access Cycles
        Slave_Cycle          ^b100100000; "

"State Machine Definitions for Simulation Only
        Sim_Idle                [Z,0,0,0,0,0,0,0,0];"
"Master Access Cycles
        Sim_PCI_Bus_Request     [Z,0,0,0,0,0,0,0,1]; "
        Sim_Drive_Address       [1,1,0,0,0,1,0,0,1]; "
        Sim_Drive_FRAME         [0,1,0,0,0,0,0,0,1]; "
        Sim_Read_Turn           [0,0,1,0,0,0,0,0,1]; "
        Sim_MDATA             = [0,0,1,0,0,0,0,1,1]; "
        Sim_Last_Burst_Xfer     [1,0,0,0,0,0,0,1,0]; "
        Sim_WaitDevice0         [0,0,0,0,1,1,0,1,0]; "
        Sim_WaitDevice1       - [0,0,0,0,1,1,1,1,0]; "
        Sim_WaitDevice2         [0,0,0,0,1,1,1,1,0]; "
        Sim_WaitDevice3       = [0,0,0,0,0,1,1,1,0]; "
        Sim_MABORT            = [1,0,0,0,0,1,0,0,0]; "
"Slave Access Cycles
        Sim_Slave_Cycle       = [Z,0,0,1,0,0,0,0,0]; "


        LDATA_Detect      = [LDATA1,IDLE_];
        DATA_Idle         = ^b00;
        Start_Trans       = 'b01;
        Last_DATA0        = 'b11;

"-------------------------------------------------------------------------------------------------
"                  PCI Controller State Machine State Diagram
"-------------------------------------------------------------------------------------------------
state_diagram PCI_Control

state Idle:
        if   !BRST_                                                    then Idle
        else if
            CYCLE_START                                               then Drive_Address
        else if
            !IBFRAME_ & LFRAME_ & IO_Read                              then Slave_Cycle
        else if
            !IBFRAME_ & LFRAME_ & IO_Write                             then Slave_Cycle
        else if
            !IBFRAME_ & LFRAME_ & Config_Read                          then Slave_Cycle
        else if
            !IBFRAME_ & LFRAME_ & Config_Write                         then Slave_Cycle
             else                                                            Idle;

state Drive_Address:
        if !BGNT_                                                      then  Drive_FRAME
        else                                                              Idle;

state Drive_FRAME:
         if  CMD_ENABLE_                                               then Idle
         else if
            PCI_WRITE                                                  then WaitDevice0
         else                                                             Read_Turn;

state Read_Turn:
        if
           !BDEVSELI_ & !DMA_SINGLE                                    then MDATA
        else if
           !BDEVSELI_ &  DMA_SINGLE                                    then Last_Burst_Xfer
        else                                                              WaitDevice1
```

```
state MDATA:
        if
            IBIRDY_ & !DMA_DONE_                                    then Last_Burst_Xfer
         else if
           !IBIRDY_ &  IBTRDY_ & !DMA_DONE_                         then Last_Burst_Xfer
        else if
           !DMA_REQ &  IBIRDY_                                      then Last_Burst_Xfer
        else if
           !BDEVSELI_ &  LDATA1                                      then Last_Burst_Xfer
        else if
            LTOVF &  BGNT_ & IBIRDY_                                 then Last_Burst_Xfer
        else if
            BDEVSELI_ & IBIRDY_                                      then Last_Burst_Xfer
        else if
           !IBSTOP_ &  !IBIRDY_                                       then Last_Burst_Xfer
        else if
           !IBSTOP_ &  IBTRDY_ & IBIRDY_                            then Last_Burst_Xfer
        else                                                            MDATA;

state Last_Burst_Xfer:
        if
           !IBTRDY_ & !IBIRDY_                                       then Idle
        else if
           !IBSTOP_                                                  then Idle
        else if
            CMD_ENABLE_                                              then Idle
        else                                                            Last_Burst_Xfer;

state WaitDevice0:
        if
           !BDEVSELI_ & !DMA_SINGLE                                 then MDATA
        else if
           !BDEVSELI_ &  DMA_SINGLE                                 then Last_Burst_Xfer
        else                                                            WaitDevice1;

state WaitDevice1:
        if
           !BDEVSELI_ & !DMA_SINGLE                                 then MDATA
        else if
           !BDEVSELI_ &  DMA_SINGLE                                 then Last_Burst_Xfer
        else                                                            WaitDevice2;

state WaitDevice2:
        if
           !BDEVSELI_ & !DMA_SINGLE                                 then MDATA
        else if
           !BDEVSELI_ &  DMA_SINGLE                                 then Last_Burst_Xfer
        else                                                            WaitDevice3;

state WaitDevice3:
        if
           !BDEVSELI_ & !DMA_SINGLE                                 then MDATA
        else if
           !BDEVSELI_ &  DMA_SINGLE                                 then Last_Burst_Xfer
        else                                                            MABORT;

state MABORT:
        if DMA_REQ                                                  then MABORT;
        else                                                            Idle;

state Slave_Cycle:
        if
           !IBIRDY_ & !IBTRDY_ & !LDEVSEL_                           then Idle"Valid Data Phase
        else if
            IBIRDY_ & IBFRAME_                                       then Idle"Master Abort
        else if
           !IBSTOP_  & !IBIRDY_                                      then Idle"Target Disconnect
        else                                                            Slave_Cycle;

"-------------------------------------------------------------------------------------------------
"             Last Data Phase Detection State Machine
"-------------------------------------------------------------------------------------------------
state_diagram LDATA_Detect
"       NOTE:  LDATA0 State and Signal Added to account for the situation
"       when !BIRDY_ and !BTRDY_ are asserted in the same clock
"       that BFRAME_ is deasserted.
state DATA_Idle:
        if !IBFRAME_ & BRST_ & IBSTOP_                                  then Start_Trans
        else                                                                DATA_Idle;

state Start_Trans:
        if !BRST_                                                       then Idle
        else if
           !IBFRAME_ & !IBSTOP_ & IBIRDY_ & IBTRDY_                     then DATA_Idle
```

```
        else if
           !IBFRAME_ &  IBSTOP_                                              then Start_Trans
        else if
            IBFRAME_ & !IBIRDY_ & !IBTRDY_                                   then DATA_Idle
        else if
            !IBSTOP_                                                         then DATA_Idle
        else if
             !ADDR_CLK                                                       then Start_Trans
        else                                                                      Last_DATA0;

state Last_DATA0:
        if !BRST_                                                            then Idle
        else if
           !IBIRDY_ & !IBTRDY_                                               then DATA_Idle
        else if
           !IBSTOP_                                                 then DATA_Idle
        else if "Cycle for other slave
            LDEVSEL_ & CMD_ENABLE_ & ADDR_CLK                                then DATA_Idle
        else if
           !CMD_ENABLE_ & (PCI_Control==MABORT)                              then DATA_Idle
        else                                                                      Last_DATA0;

"-------------------------------------------------------------------------------------------------
"                 Equations
"-------------------------------------------------------------------------------------------------
equations
        BRST_.OE   1;
        BRST_.CLK   BCLK;
        BRST_ := LBRST_;
"-------------------------------------------------------------------------------------------------
"        State Machine Clock Equations and Register Output Enable Equations
"-------------------------------------------------------------------------------------------------
         MAS0.CLK              = BCLK;
         MAS0.OE               = 1;
         MAS0.AR               = !BRST_;
         MAS1.CLK              = BCLK;
         MAS1.OE               = 1;
         MAS1.AR                 !BRST_;
         MAS2.CLK                BCLK;
         MAS2.OE                 1;
         MAS2.AR                 !BRST_;
         MAS3.CLK                BCLK;
         MAS3.OE                 1;
         MAS3.AR               = !BRST_;
         MAS4.CLK              = BCLK;
         MAS4.OE               = 1;
         MAS4.AR                 !BRST_;
         MAS5.CLK                BCLK;
         MAS5.OE                 1;
         MAS5.AR                 !BRST_;
         PCI_ADEN_.CLK           BCLK;
         PCI_ADEN_.OE            1;
         PCI_ADEN_.AR            !BRST_;
         ADDR_OE.CLK             BCLK;
         ADDR_OE.OE              1;
         ADDR_OE.AR              !BRST_;
         BFRAME_.CLK              BCLK;
         BFRAME_.ASET            !BRST_;
         BFRAME_.OE              !CMD_ENABLE_;

         LDATA1.CLK              BCLK;
         LDATA1.OE               1;
         IDLE_.CLK               BCLK;
         IDLE_.OE                1;

"-------------------------------------------------------------------------------
"       On-board FIFO Read Control Logic
"-------------------------------------------------------------------------------
         FIFO_RD_ENA.OE           SIMULATE;
         FIFO_RD_ENA.CLK          BCLK;
         FIFO_RD_ENA           := !CMD_ENABLE_ & !ADDR_OE & PCI_WRITE & IDLE_;

         IFIFO_RD_                !CMD_ENABLE_ &  FIFO_GBA & !IBIRDY_ & !IBTRDY_ & !PCI_CLKIN & DMA_DONE_S_
                                #  SLAVE_FIFO_RD_ & CMD_ENABLE_ & DMA_DONE_S_ & !CHAIN;

"-------------------------------------------------------------------------------
"       Parity Generation and Error Detection Logic
"-------------------------------------------------------------------------------

         PARITY_CHECK.OE    1;
         PARITY_CHECK.AR  = !BRST_;
         PARITY_CHECK.CLK = BCLK;
         PARITY_CHECK := !IBIRDY_ & !IBTRDY_;
```

```
        IPAR_.OE      0;
        IPAR_.CLK     BCLK;
        IPAR_      :=  PARO_3 $ PARO_2 $ PARO_1 $ PARO_0;

        BPAR_.OE      PCI_ADDIR_D;
        BPAR_.CLK     BCLK;
        BPAR_      :=  PARO_3 $ PARO_2 $ PARO_1 $ PARO_0;

        BPERR_.OE      1;
        BPERR_.CLK     BCLK;
        BPERR_.ASET   !BRST_;
       !BPERR_      := (IPAR_ $  BPARIN_) & !CMD_ENABLE_ & PARITY_CHECK & !PCI_WRITE   "Master Read
                     # (BPAR_ $  BPARIN_) &  CMD_ENABLE_ & PARITY_CHECK &  PCI_WRITE;  "Slave Write

        SERR_.OE   1;
        SERR_.CLK = BCLK;
        SERR_.ASET   !BRST_;
        SERR_      := 1;
"       !SERR_     := !BPERR_ & SYSERR_ENABLE;

        LSERR.OE    1;
        LSERR.CLK   BCLK;
        LSERR.AR    !BRST_;
        LSERR      := !SERR_
                   #  LSERR & !STATUS_STROBE_SYN;

"        BC3L.OE          1;
"        BC3L.CLK         BCLK;
"        BC3L.D       :=
                        "CBC3.FB &  ADDR_PHASE & (DMACycle == MemRDLine)
                       "#  CBC3.FB &  ADDR_PHASE & (DMACycle == MemWRInv)
"                         BC3L &  BC2L & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC3L &  BC1L & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC3L &  BC0L & !ADDR_PHASE & !CMD_ENABLE_
"                       # !BC3L & !BC2L & !BC1L & !BC0L & !IBTRDY_ & !CMD_ENABLE_
"                       #  BC3L &  IBTRDY_ & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC3L & CMD_ENABLE_;

"        BC2L.OE          1;
"        BC2L.CLK         BCLK;
"        BC2L.D       :=
                        "CBC2.FB &  ADDR_PHASE & (DMACycle == MemRDLine)
                       "#  CBC2.FB &  ADDR_PHASE & (DMACycle == MemWRInv)
"                         BC2L &  BC1L & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC2L &  BC0L & !ADDR_PHASE & !CMD_ENABLE_
"                       # !BC2L & !BC1L & !BC0L & !IBTRDY_ & !CMD_ENABLE_
"                       #  BC2L & IBTRDY_ & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC2L & CMD_ENABLE_;

"        BC1L.OE      =  1;
"        BC1L.CLK     =  BCLK;
"        BC1L.D       :=
                        "CBC1.FB &  ADDR_PHASE & (DMACycle == MemRDLine)
                       "#  CBC1.FB &  ADDR_PHASE & (DMACycle == MemWRInv)
"                         BC1L &  BC0L & !ADDR_PHASE & !CMD_ENABLE_
"                       # !BC1L & !BC0L & !IBTRDY_ & !CMD_ENABLE_
"                       #  BC1L &  IBTRDY_ & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC1L & CMD_ENABLE_;

"        BC0L.OE          1;
"        BC0L.CLK         BCLK;
"        BC0L.D       :=
                       "CBC0.FB &  ADDR_PHASE & (DMACycle == MemRDLine)
                       "#  CBC0.FB &  ADDR_PHASE & (DMACycle == MemWRInv)
"                        !BC0L & !IBTRDY_ & !CMD_ENABLE_
"                       #  BC0L &  IBTRDY_ & !ADDR_PHASE & !CMD_ENABLE_
"                       #  BC0L & CMD_ENABLE_;

"        BURST_DONE.OE  =  1;
"        BURST_DONE.CLK =  BCLK;
"        BURST_DONE     = !BC3L.FB & !BC2L.FB & BC1L.FB & !BC0L.FB & DMA_S3 & DMA_S1
"                         # !BC3L.FB & !BC2L.FB & BC1L.FB &  BC0L.FB & DMA_S3 & DMA_S1
"                                    & !IBIRDY_ & !IBTRDY_
"                         # BURST_DONE & !CMD_ENABLE_;

        PCI_WRITE.OE  =  1;
        PCI_WRITE.CLK    BCLK;
        PCI_WRITE     := !CMD_ENABLE_ & (DMACycle == MemWR) & !CHAIN"Master Write
                       #  CMD_ENABLE_ & !IBFRAME_ & IO_Write "Slave IO Write
                       #  CMD_ENABLE_ & !IBFRAME_ & (PCI_Cmd==Config_Write)
                       #  CMD_ENABLE_ & IDLE_ & PCI_WRITE;    "Hold until Idle
"-------------------------------------------------------------------------------
"       PCI Bus Signal Control Logic
"-------------------------------------------------------------------------------
```

```
          BSTOP_.OE    =  CMD_ENABLE_;
          BSTOP_          1;


         DMA_SINGLE     DMA_BREQUEST_ & !DMA_REQUEST_
                      # !DMA_MODE2 & !DMA_MODE1 &  DMA_MODE0   "Fly By Count Down
                      # !DMA_MODE2 &  DMA_MODE1 &  DMA_MODE0;  "Chain Count Down

         BTRDY_.OE       CMD_ENABLE_;
         BTRDY_.CLK      BCLK;
         BTRDY_.ASET    !BRST_;
        !BTRDY_      :=  IBTRDY    &  !LDEVSEL_ & IDLE_ & LDATA1 & IBSTOP_ & !BTRDY_D & !BTRDY_DD &
!BTRDY_DDD
                      # !IBTRDY_ &  IBIRDY_ & IBSTOP_ &  CMD_ENABLE_;

         BTRDY_D.OE      0;
         BTRDY_D.CLK     BCLK;
         BTRDY_D.ASET = !BRST_;
        !BTRDY_D      := !LDEVSEL_ & IDLE_ & LDATA1 & IBSTOP_;

         BTRDY_DD.OE   = 0;
         BTRDY_DD.CLK    BCLK;
         BTRDY_DD.ASET   !BRST_;
         !BTRDY_DD     := !LDEVSEL_ & IDLE_ & LDATA1 & IBSTOP_ & !BTRDY_D;

          BTRDY_DDD.OE   = 0;
          BTRDY_DDD.CLK  = BCLK;
          BTRDY_DDD.ASET = !BRST_;
         !BTRDY_DDD      := !LDEVSEL_ & IDLE_ & LDATA1 & IBSTOP_ & !BTRDY_DD;

         BREQ_.OE        1;
         BREQ_.CLK       BCLK;
         BREQ_.ASET      !BRST_;
        !BREQ_        :=  DMA_REQ & IBSTOP_ & !LTOVF &  DMA_MODE1   "BREQ_ Must be deassserted for 2 clocks
                      #   DMA_REQ & IBSTOP_ & !LTOVF &  DMA_DONE_ & !DMA_MODE1
                      #   DMA_REQ & IBSTOP_ & !BGNT_ &  DMA_MODE1
                      #   DMA_REQ & IBSTOP_ & !BGNT_ &  DMA_DONE_ & !DMA_MODE1;

         BIRDY_.CLK      BCLK;
         BIRDY_.OE      !CMD_ENABLE_;

        !BIRDY_       :=  IBIRDY_ & !CMD_ENABLE_ & !IBTRDY_       "Master Writes Wait for TRDY for FIFOFL
Strobe
                      #   IBIRDY_ & !CMD_ENABLE_ & !BDEVSELI_ & !IBFRAME_ & IBSTOP_
                      #  !IBIRDY_ & !CMD_ENABLE_ &  IBTRDY_ & IBSTOP_ & !BDEVSELI_
                      #  !IBIRDY_ & !CMD_ENABLE_ & !IBFRAME_ &  IBTRDY_
                      #  !IBIRDY_ & !IBSTOP_
                      #   BIRDY_ & (LDATA_Detect == Last_DATA0) & BTRDY_ & !CMD_ENABLE_;

         MASTER_.OE    =  1;
         MASTER_.CLK   =  BCLK;
         MASTER_.ASET  = !BRST_;
        !MASTER_      := !CMD_ENABLE_ &  IBIRDY_ & LFRAME_ & MASTER_    "Turn on 1 clock after CMD
                       # !MASTER_ &  IDLE_   "Maintain Until End of Transaction
                       # !MASTER_ & !IBIRDY_   "Maintain Until End of Transaction
                       # !MASTER_ & !CMD_ENABLE_ & !IBFRAME_; "Turn off after FRAME

         BC_BE3_.OE     !CMD_ENABLE_;
        !BC_BE3_        !CMD_ENABLE_ &  IDLE_ "Active for Data Transfers
                      # !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemRD) "Active for Mem Read
                      # !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemWR) "Active for Mem Write
                      # !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemWRInv);
                                                "Active for Mem Write and Invalidate if Enabled

         BC_BE2_.OE     !CMD_ENABLE_;
        !BC_BE2_        !CMD_ENABLE_ &  IDLE_; " Only Active for Data  Transfers

         BC_BE1_.OE     !CMD_ENABLE_;
        !BC_BE1_        !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemRDMult)"Active for Mem Read Mult
                      # !CMD_ENABLE_ &  IDLE_;       "Active for Data Transfers

         BC_BE0_.OE     !CMD_ENABLE_;
        !BC_BE0_        !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemRD)
                      # !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemRDMult)
                      # !CMD_ENABLE_ & !IDLE_ & (DMACycle == MemRDLine)  "Active for All Mem Read CMDs
                      #  CHAIN                       "Active in Chaining Mode
                      # !CMD_ENABLE_ &  IDLE_;       "Active for Data Transfers

        LFRAME_.OE    =  1;
        LFRAME_.ASET  = !BRST_;
        LFRAME_.CLK   =  BCLK;
       !LFRAME_       := !IBFRAME_
                       # !LFRAME_.FB &  IDLE_;  "Hold until PCI Bus Idle
```

```
        BDEVSEL_.OE      CMD_ENABLE_;
       !BDEVSEL_        !LDEVSEL_ & (PCI_Control==Slave_Cycle);

        "Assert INTA_ Active low upon DMA_INTRQ being asserted, Clear on CLS Write
        INTA_.OE     = 1;
        INTA_.CLK       DMA_INTRQ & BRST_;
        INTA_.ASET      !CLS_WRITE_ # !BRST_;
        INTA_         := 0;

        "The CMD_ENABLE_ signal signifies that we have been granted the
        "PCI Bus and will begin an initiator cycle.  This signal
        "envelopes the entire initiator cycle.
        CMD_ENABLE_.OE     1;
        CMD_ENABLE_.CLK    BCLK;
        CMD_ENABLE_.ASET = !BRST_;
        !CMD_ENABLE_     :=  CMD_ENABLE_ & CYCLE_START & !BGNT_
                         # !CMD_ENABLE_ &  DMA_REQ & (PCI_Control == Drive_Address)
                         # !CMD_ENABLE_ & !IBFRAME_
                         # !CMD_ENABLE_ & !IBIRDY_
                         # !CMD_ENABLE_ &  IBFRAME_ & (LDATA_Detect != DATA_Idle)
                         # !CMD_ENABLE_ &  IBFRAME_ & (LDATA_Detect == Last_DATA0)
                                        &  IBTRDY_ & IBIRDY_;

        CYCLE_START.OE     1;
        CYCLE_START.CLK    BCLK;
        CYCLE_START    :=  (PCI_Control==Idle) & !BGNT_ & DMA_REQ & MASTER_ENABLE &  IBFRAME_ & !IDLE_ &
LFRAME_ & LDEVSEL_;

       !PCI_ADDIR          CMD_ENABLE_ & !PCI_WRITE & !LDEVSEL_ & !LFRAME_ & !BDEVSELI_ "Slave Reads
                        # !CMD_ENABLE_ & !BFRAME_ &   BDEVSELI_
                        # !CMD_ENABLE_ &  PCI_WRITE & IDLE_
                        #  ADDR_OE;

        PCI_ADDIR_D.OE    1;
        PCI_ADDIR_D.CLK   BCLK;
        PCI_ADDIR_D   :=  PCI_ADDIR;

        PCI_ADEN_D.OE     1;
        PCI_ADEN_D.CLK -  BCLK;
        PCI_ADEN_D.ASET   !BRST_;
        PCI_ADEN_D    :=  1;
"        PCI_ADEN_D    :=  PCI_ADEN_;

        STORE_DATAI.OE     1;
        STORE_DATAI.CLK   !BCLK;
        STORE_DATAI    :=  DMA_REQ & !MASTER_ & IDLE_ & !PCI_WRITE  "Master Read
                        # !MASTER_ &  STORE_DATAI                    "& !IBIRDY_             Hold till
last cycle
                        #  MASTER_ &  ADDR_CLK                      "New added for Address In Phase
                        #  MASTER_                                  "Slave Cycles
                        #  MASTER_ & !LFRAME_ & PCI_WRITE;          "Slave Write

        STORE_DATAO.OE   =  1;
        STORE_DATAO.CLK  =  BCLK;
        STORE_DATAO     := !CMD_ENABLE_ & ADDR_OE;

        STORE_CLKO.OE    =  1;
        STORE_CLKO.CLK      BCLK;
        STORE_CLKO      :=  ADDR_OE;

        DMA_ACK.OE          1;
        DMA_ACK.CLK         BCLK;
        DMA_ACK         := !CMD_ENABLE_ & !IBIRDY_ & !IBTRDY_;

        ADDR_PHASE.OE    1;
        ADDR_PHASE.CLK   BCLK;
        ADDR_PHASE    := !CMD_ENABLE_ & MASTER_
                        #  CMD_ENABLE_ & MASTER_ & !IBFRAME_ &  LFRAME_;

        ADDR_CLK.OE     1;
        ADDR_CLK.CLK = !BCLK;
        ADDR_CLK    := !LFRAME_;

"-------------------------------------------------------------------------------------------
"           PCI Status Output Latches
"-------------------------------------------------------------------------------------------

        STATUS_STROBE_SYN.OE     1;
        STATUS_STROBE_SYN.CLK    BCLK;
        STATUS_STROBE_SYN    :=  STATUS_STROBE;

        MPARER_RPT.OE    1;
        MPARER_RPT.CLK   BCLK;
        MPARER_RPT    := !MASTER_ & !BPERR_ & PARERR_ENABLE
```

```
                    #  MPARER_RPT & .STATUS_STROBE_SYN;
        MPARER_RPT.AR     !BRST_;

        MTRG_ABORT.OE  = 1;
        MTRG_ABORT.CLK = BCLK;
        MTRG_ABORT     := !CMD_ENABLE_ & .IRDY_ & BDEVSEL_
                        #  MTRG_ABORT & .STATUS_STROBE_SYN;
        MTRG_ABORT.AR     !BRST_;

        STRG_ABORT.OE     1;
        STRG_ABORT.CLK    BCLK;
        STRG_ABORT     :=  CMD_ENABLE_ & .DEVSEL_ & .IRDY_
                        #  STRG_ABORT & !STATUS_STROBE_SYN;
        STRG_ABORT.AR     .BRST_;

        MASTER_ABORT.OE    1;
        MASTER_ABORT.CLK   BCLK;
        MASTER_ABORT     := BDEVSEL_ & (PCI_Control == WaitDevice3)
                          # MASTER_ABORT & !STATUS_STROBE_SYN;
        MASTER_ABORT.AR    .BRST_;

        ABAPER.OE      =  1;
        ABAPER.CLK     =  BCLK;
        ABAPER         := .DEVSEL_ & BRST_
                        #  ABAPER &  .STATUS_STROBE_SYN;
        ABAPER.AR         .BRST_;

end PCICTRL
```

5.3.2. DMAC PLD

```
module DMAC
title 'DMA PCI Master Controller Logic
      for the PCI Interface Board
Sheet 2 of 10, Schematic Diagrams
"-------------------------------------------------------------------------------------
"               PCI Direct Memory Access Controller
"-------------------------------------------------------------------------------------
"-------------------------------------------------------------------------
"
"  This FPGA implements the following functionality on the PCI
"  PCI I/O board:
"
"    Initiates Direct Memory Access Cycles as a PCI Master when required by
"    the Parallel Interface.
"
"    Acts as the Address Generation Control Logic for all Master PCI Cycles.
"    This includes the register functionality for receiving the initial DMA
"    address from another PCI Master, properly incrementing address with
"    each successful DMA Cycle, and Driving Address information during the
"    Address Phase of each PCI Master Cycle.
"
"    Performs Address Generation and I/O Selection for 32-bit aligned
"    fly-by or single cycle DMA Transfers between system memory and the
"    on-board FIFO memory buffer via the DMA State Machine.
"
"    Performs Address Generation for dual cycle memory-to-memory DMA
"    Transfers between two system memory locations.
"
"-------------------------------------------------------------------------------------
"                 DMA Control Register Inputs
"-------------------------------------------------------------------------------------
"     The least significant 16-bits of the DMA Control Register are inputs
"     to this FPGA.  They are READ/WRITE accessible by a PCI Master.
"     Currently Bits 15-10 are undefined and therefore not described here.
"
"     Bits 9 and 8 determine the Interrupt Mode Selected as follows:
"     ---------------------------------------------------------------------
"     |         Mode 1(9)     Mode 0(8)                                    |
"     |-------------------------------------------------------------------|
"     |           0             0        No Interrupt Generated           |
"     |           0             1        Interrupt on Terminal Count      |
"     |           1             0        Interrupt Only on END OF Last Chain|
"     |           1             1        No Interrupt Generated.          |
"     |___________________________________________________________________|
"
"
"     Bits 7, 6,  and 5 determine the DMA Transfer Mode Selected as follows:
"    ---------------------------------------------------------------------
"    | Mode 2(7)   Mode 1(6)     Mode 0(5)                                |
"    |-------------------------------------------------------------------|
"    |   0           0             0     Fly-by Count Up in Normal Mode   |
"    |   0           0             1     Fly-by Count Down in Normal Mode |
"    |   0           1             0     Fly-by Count Up in Chaining Mode |
"    |   0           1             1     Fly-by Count Down in Chaining Mode|
"    |   1           0             0     Mem-mem Count Up in Normal Mode  |
"    |   1           0             1     Mem-mem Count Down in Normal Mode|
"    |   1           1             0     Invalid Combination              |
"    |   1           1             1     Invalid Combination              |
"    |___________________________________________________________________|
"
"
"     Bits 4,3,2, and 1 Describe the PCI Cycle Type as follows:
"     ------------------------------------------------------------
"     |          S3   S2   S1   S0                                |
"     |-----------------------------------------------------------|
"     |           0    1    1    0       Memory Read             |
"     |           0    1    1    1       Memory Write            |
"     |           1    1    0    0       Memory Read Multiple    |
"     |           1    1    1    0       Memory Read Line        |
"     |           1    1    1    1       Memory Write & Invalidate|
"     |___________________________________________________________|
"     Note that these 4 Mode bits not only determine the Cycle type, but they
"     determine the direction of the transfer.  All transfer directions are
"     with respect to the PCI Bus.  For example a Memory Read indicates that
"     a READ will take place across the PCI Bus, and a write will take place
"     to the local resource.  Likewise, a Memory Write & Invalidate indicates
"     that the data transfer direction will be from the board to a system
"     memory resource on the PCI Bus.
"
```

```
"-------------------------------------------------------------------------------
"                  Declarations
"-------------------------------------------------------------------------------

                   DMAC    device 'ifx780fp' ;

"-------------------------------------------------------------------------------
"                  Inputs Pins
"-------------------------------------------------------------------------------
        BCLK                    pin  118;" 33Mhz PCI Bus Clock
        BRST_                   pin    1;"  Buffered Active Low PCI Reset
        ADDR_PHASE              pin    2;"  PCI Address Phase Detect
        DATA_PHASE_             pin    3;"  PCI Data Phase Detect#
        MASTER_                 pin    4;"  PCI Bus Master Status#
        DMA_SOURCE_WR_          pin    5;"  DMA Source Address Write Strobe#
        DMA_SOURCE_RD_          pin   33;"  DMA Source Address Read Strobe#
        DMA_STAT_RD_            pin   42;"  DMA Status Read Strobe#
        ADDR_OE                 pin   34;"  PCI Master Address Output Enable
        DMA_GRANT_              pin   35;"  DMA Transfer Grant#
        DMA_ACK                 pin   36;"  DMA Transfer Acknowledge
        DMA_START               pin   37;"  DMA Start Command Signal
        DMA_BREQUEST_           pin   38;"  DMA Burst Request#
        DMA_REQUEST_            pin   67;"  DMA Single Request#
        DMA_DONE_               pin   68;"  DMA Done From Longword Transfer Counter#
        CHAIN_OVF_              pin  111;" Chaining Register Overflow#
        CHAIN_OVF0_             node  28;"  Chaining Register Overflow 0#
        CHAIN_OVF1_             node  48;"  Chaining Register Overflow 1#
        CHAIN_OVF2_             node  63;"  Chaining Register Overflow 2#
        CHAIN_OVF3_             node  90;"  Chaining Register Overflow 3#
        CHAIN_OVF4_             node  15;"  Chaining Register Overflow 4#

        DMA_COUNT_WR_           pin   70;"  DMA Count Write Input#
        DMA_CHAIN_RD_           pin   71;"  DMA Chaining Register Read Strobe#
        DMA_CMAIN_WR_           pin  109;"  DMA Chaining Register Write Strobe#

        TC_STAGE1_              pin  114; "Terminal Count Stage 1 Out

        CLS_WRITE_              pin 123 ; "INT Clear Strobe

"-------------------------------------------------------------------------------
"       DMA Control Register Inputs
"-------------------------------------------------------------------------------
        INT_MODE1               pin   99;"  DMA Control Register Bit 9 (Interrupt Mode 1)
        INT_MODE0               pin  100;"  DMA Control Register Bit 8 (Interrupt Mode 0)
        DMA_MODE2               pin  101;"  DMA Control Register Bit 7 (DMA Mode 2)
        DMA_MODE1               pin  102;"  DMA Control Register Bit 6 (DMA Mode 1)
        DMA_MODE0               pin  103;"  DMA Control Register Bit 5 (DMA Mode 0)
        DMA_S0                  pin   62;   "DMA Control Register Bit 1 (Cycle Type S0)

"-------------------------------------------------------------------------------
"                  Bi-Directional Pins
"-------------------------------------------------------------------------------
        LOADCTRL                pin   91; "DMA Master State Machine Load Control Word
        CHAIN                   pin  122; "DMA Master State Machine CMAIN
        S2                      node  128;"  DMA Master State Machine S2
        S1                      NODE  123;"  DMA Master State Machine S1
        S0                      node 127;"  DMA Master State Machine S0
        DMA_INTRQ               pin  129;  "DMA Controller Interrupt Request

        PCICLKIN_D              node  69;"  PCI Input Clock used for Equations
        PCICLKIN                pin   57;  "PCI Input Clock used for Equations

        BADDR31                 pin  22 ; "Buffered Address Value 31
        BADDR30                 pin  23 ; "Buffered Address Value 30
        BADDR29                 pin  24 ; "Buffered Address Value 29
        BADDR28                 pin  25 ; "Buffered Address Value 28
        BADDR27                 pin   6 ; "Buffered Address Value 27
        BADDR26                 pin  98 ; "Buffered Address Value 26
        BADDR25                 pin  29 ; "Buffered Address Value 25
        BADDR24                 pin  30 ; "Buffered Address Value 24
        BADDR23                 pin  39 ; "Buffered Address Value 23
        BADDR22                 pin  40 ; "Buffered Address Value 22
        BADDR21                 pin  41 ; "Buffered Address Value 21
        BADDR20                 pin  97 ; "Buffered Address Value 20
        BADDR19                 pin  43 ; "Buffered Address Value 19
        BADDR18                 pin  45 ; "Buffered Address Value 18
        BADDR17                 pin   7 ; "Buffered Address Value 17
        BADDR16                 pin  47 ; "Buffered Address Value 16
        BADDR15                 pin  54 ; "Buffered Address Value 15
        BADDR14                 pin  55 ; "Buffered Address Value 14
        BADDR13                 pin  56 , "Buffered Address Value 13
        BADDR12                 pin   8 , "Buffered Address Value 12
        BADDR11                 pin  58 ; "Buffered Address Value 11
        BADDR10                 pin  60 ; "Buffered Address Value 10
```

```
        BADDR9                  pin  61 ; "Buffered Address Value 9
        BADDR8                  pin  96 ; "Buffered Address Value 8
        BADDR7                  pin   72 ; "Buffered Address Value 7
        BADDR6                  pin   73 ; "Buffered Address Value 6
        BADDR5                  pin   74 ; "Buffered Address Value 5
        BADDR4                  pin   75 ; "Buffered Address Value 4
        BADDR3                  pin   76 ; "Buffered Address Value 3
        BADDR2                  pin    9 ; "Buffered Address Value 2
        BADDR1                  pin   79 ; "Buffered Address Value 1
        BADDR0                  pin   80 ; "Buffered Address Value 0

        BAD31                   pin   22 ; "Buffered Address/Data Bit 31
        BAD30                   pin   23 ; "Buffered Address/Data Bit 30
        BAD29                   pin   24 ; "Buffered Address/Data Bit 29
        BAD28                   pin   25 , "Buffered Address/Data Bit 28
        BAD27                   pin    6 , "Buffered Address/Data Bit 27
        BAD26                   pin   98 , "Buffered Address/Data Bit 26
        BAD25                   pin   29 , "Buffered Address/Data Bit 25
        BAD24                   pin   30 , "Buffered Address/Data Bit 24
        BAD23                   pin   39 , "Buffered Address/Data Bit 23
        BAD22                   pin   40 , "Buffered Address/Data Bit 22
        BAD21                   pin   41 ; "Buffered Address/Data Bit 21
        BAD20                   pin   97 ; "Buffered Address/Data Bit 20
        BAD19                   pin   43 ; "Buffered Address/Data Bit 19
        BAD18                   pin   45 ; "Buffered Address/Data Bit 18
        BAD17                   pin    7 ; "Buffered Address/Data Bit 17
        BAD16                   pin   47 ; "Buffered Address/Data Bit 16
        BAD15                   pin   54 ; "Buffered Address/Data Bit 15
        BAD14                   pin   55 ; "Buffered Address/Data Bit 14
        BAD13                   pin   56 ; "Buffered Address/Data Bit 13
        BAD12                   pin    8 ; "Buffered Address/Data Bit 12
        BAD11                   pin   58 ; "Buffered Address/Data Bit 11
        BAD10                   pin   60 ; "Buffered Address/Data Bit 10
        BAD9                    pin   61 ; "Buffered Address/Data Bit 9
        BAD8                    pin   96 ; "Buffered Address/Data Bit 8
        BAD7                    pin   72 ; "Buffered Address/Data Bit 7
        BAD6                    pin   73 , "Buffered Address/Data Bit 6
        BAD5                    pin   74 , "Buffered Address/Data Bit 5
        BAD4                    pin   75 , "Buffered Address/Data Bit 4
        BAD3                    pin   76 , "Buffered Address/Data Bit 3
        BAD2                    pin    9 , "Buffered Address/Data Bit 2

        CH_AD7                  node  31; "Chain Address Register Bit 7
        CH_AD6                  node  49; "Chain Address Register Bit 6
        CH_AD5                  node  26; "Chain Address Register Bit 5
        CH_AD4                  node  46; "Chain Address Register Bit 4
        CH_AD3                  node  42; "Chain Address Register Bit 3
        CH_AD2                  node  32; "Chain Address Register Bit 2

        CARRY_                  pin   78; "Address Counter Overflow#
        ADR7                    node  82; "Address Counter Bit 7
        ADR6                    node 115; "Address Counter Bit 6
        ADR5                    node 114; "Address Counter Bit 5
        ADR4                    node 107; "Address Counter Bit 4
        ADR3                    node  16; "Address Counter Bit 3
        ADR2                    node  62; "Address Counter Bit 2

"-----------------------------------------------------------------------------------------------
"                 Outputs
"-----------------------------------------------------------------------------------------------

        DMA_REQ                 pin 120;  "DMA Cycle Request#
        DMA_COUNT_LOAD_         pin 130;  "DMA Transfer Count Load#
        DMA_COUNT_ENA_          pin 106;  "DMA Transfer Count Enable#
        DMA_DONE_S_             pin 108;  "DMA Transfer Count Done Sync#

        DMA_CHAIN_OE_           pin 121; "DMA Chain Output Enable#

        DMA_ADDR_ENA_           pin  94;  "DMA Address Counter Enable#
        DMA_ADDR_CLK            pin  95;  "DMA Address Counter Clock
        DMA_ADDR_LOAD_          pin 126;  "DMA Address Load Strobe#
        DMA_ADDR_OE_            pin  92;  "DMA Address Output Enable#

        DMA_DIR                 pin 105;  "DMA Direction Status Output

        DONE_BRST_               node ; "Async Reset Term For DMA State Machine

        TC_STAGE1_S_             pin 13 ; "Terminal Count Stage 1 Synchronized

"-----------------------------------------------------------------------------------------------
"                 Set Definitions and Constants
"-----------------------------------------------------------------------------------------------

        H,L,X,C,K,Z,SIMULATE = 1,0,.X.,.C.,.K.,.Z.,0;
```

```
LOADCTRL  istype  'buffer,reg,pos';
CHAIN     istype  'buffer,reg,pos';
S2        istype  'buffer,reg,pos';
S1        istype  'buffer,reg,pos';
S0        istype  'buffer,reg,pos';
DMA_REQ   istype  'buffer,reg,pos';


BADDR31   istype  'buffer,reg,pos';
BADDR30   istype  'buffer,reg,pos';
BADDR29   istype  'buffer,reg,pos';
BADDR28   istype  'buffer,reg,pos';
BADDR27   istype  'buffer,reg,pos';
BADDR26   istype  'buffer,reg,pos';
BADDR25   istype  'buffer,reg,pos';
BADDR24   istype  'buffer,reg,pos';
BADDR23   istype  'buffer,reg,pos';
BADDR22   istype  'buffer,reg,pos';
BADDR21   istype  'buffer,reg,pos';
BADDR20   istype  'buffer,reg,pos';
BADDR19   istype  'buffer,reg,pos';
BADDR18   istype  'buffer,reg,pos';
BADDR17   istype  'buffer,reg,pos';
BADDR16   istype  'buffer,reg,pos';
BADDR15   istype  'buffer,reg,pos';
BADDR14   istype  'buffer,reg,pos';
BADDR13   istype  'buffer,reg,pos';
BADDR12   istype  'buffer,reg,pos';
BADDR11   istype  'buffer,reg,pos';
BADDR10   istype  'buffer,reg,pos';
BADDR9    istype  'buffer,reg,pos';
BADDR8    istype  'buffer,reg,pos';
BADDR7    istype  'com';
BADDR6    istype  'com';
BADDR5    istype  'com';
BADDR4    istype  'com';
BADDR3    istype  'com';
BADDR2    istype  'com';
BADDR1    istype  'com';
BADDR0    istype  'com';

CH_AD7    istype  'buffer,reg,pos';
CH_AD6    istype  'buffer,reg,pos';
CH_AD5    istype  'buffer,reg,pos';
CH_AD4    istype  'buffer,reg,pos';
CH_AD3    istype  'buffer,reg,pos';
CH_AD2    istype  'buffer,reg,pos';

ADR7      istype  'buffer,reg,pos';
ADR6      istype  'buffer,reg,pos';
ADR5      istype  'buffer,reg,pos';
ADR4      istype  'buffer,reg,pos';
ADR3      istype  'buffer,reg,pos';
ADR2      istype  'buffer,reg,pos';

Cur_Addr_B0       = [ADR7.Q ,ADR6.Q ,ADR5.Q ,ADR4.Q ,
                     ADR3.Q ,ADR2.Q];

Chain_Addr_B0     = [CH_AD7.Q ,CH_AD6.Q ,CH_AD5.Q ,CH_AD4.Q ,
                     CH_AD3.Q ,CH_AD2.Q];

Cur_Addr          = [BADDR31.Q,BADDR30.Q,BADDR29.Q,BADDR28.Q,BADDR27.Q,
                     BADDR26.Q,BADDR25.Q,BADDR24.Q,BADDR23.Q,BADDR22.Q,
                     BADDR21.Q,BADDR20.Q,BADDR19.Q,BADDR18.Q,BADDR17.Q,
                     BADDR16.Q,BADDR15.Q,BADDR14.Q,BADDR13.Q,BADDR12.Q,
                     BADDR11.Q,BADDR10.Q,BADDR9.Q ,BADDR8.Q ,CH_AD7.Q ,
                     CH_AD6.Q ,CH_AD5.Q ,CH_AD4.Q ,CH_AD3.Q ,CH_AD2.Q];

Cur_Addr_Next_B0  [ADR7.D ,ADR6.D ,ADR5.D ,ADR4.D ,
                   ADR3.D ,ADR2.D];

Chain_Addr_Next_B0 = [CH_AD7.D ,CH_AD6.D ,CH_AD5.D ,CH_AD4.D ,
                      CH_AD3.D ,CH_AD2.D];

Cur_Addr_Next     [BADDR31.D,BADDR30.D,BADDR29.D,BADDR28.D,BADDR27.D,
                   BADDR26.D,BADDR25.D,BADDR24.D,BADDR23.D,BADDR22.D,
                   BADDR21.D,BADDR20.D,BADDR19.D,BADDR18.D,BADDR17.D,
                   BADDR16.D,BADDR15.D,BADDR14.D,BADDR13.D,BADDR12.D,
                   BADDR11.D,BADDR10.D,BADDR9.D ,BADDR8.D ,CH_AD7.D ,
                   CH_AD6.D ,CH_AD5.D ,CH_AD4.D ,CH_AD3.D ,CH_AD2.D];

Addr_In_B0        [BAD7,BAD6,BAD5,BAD4,BAD3,BAD2];
```

```
        Addr_In             [BAD31,BAD30,BAD29,BAD28,BAD27,
                             BAD26,BAD25,BAD24,BAD23,BAD22,
                             BAD21,BAD20,BAD19,BAD18,BAD17,
                             BAD16,BAD15,BAD14,BAD13,BAD12,
                             BAD11,BAD10,BAD9 ,BAD8 ,BAD7 ,
                             BAD6 ,BAD5 ,BAD4 ,BAD3 ,BAD2];

        Addr_In_All         [BAD31,BAD30,BAD29,BAD28,BAD27,
                             BAD26,BAD25,BAD24,BAD23,BAD22,
                             BAD21,BAD20,BAD19,BAD18,BAD17,
                             BAD16,BAD15,BAD14,BAD13,BAD12,
                             BAD11,BAD10,BAD9 ,BAD8 ,BAD7 ,
                             BAD6 ,BAD5 ,BAD4 ,BAD3 ,BAD2 ,
                             BADDR1,BADDR0];

        Addr_Out            [BADDR31,BADDR30,BADDR29,BADDR28,BADDR27,
                             BADDR26,BADDR25,BADDR24,BADDR23,BADDR22,
                             BADDR21,BADDR20,BADDR19,BADDR18,BADDR17,
                             BADDR16,BADDR15,BADDR14,BADDR13,BADDR12,
                             BADDR11,BADDR10,BADDR9 ,BADDR8 ,BADDR7 ,
                             BADDR6 ,BADDR5 ,BADDR4 ,BADDR3 ,BADDR2];

        Addr_Out_All        [BADDR31,BADDR30,BADDR29,BADDR28,BADDR27,
                             BADDR26,BADDR25,BADDR24,BADDR23,BADDR22,
                             BADDR21,BADDR20,BADDR19,BADDR18,BADDR17,
                             BADDR16,BADDR15,BADDR14,BADDR13,BADDR12,
                             BADDR11,BADDR11,BADDR10,BADDR9 ,BADDR8];

        Addr_Out_B0         [BADDR7 ,BADDR6 ,BADDR5 ,BADDR4 ,BADDR3 ,
                             BADDR2 ,BADDR1 , BADDR0];


         DMA_Mode                [ DMA_MODE1];
         FlyBy_Up_Normal         [    0     ];
         FlyBy_Up_Chain          [    1     ];

        Chain_Mode      DMA_MODE1;
        Normal_Mode    !DMA_MODE1;

        Count_Up      1;

"-------------------------------------------------------------------------------------
"                  DMA State Machine State Definition
"-------------------------------------------------------------------------------------
        DMA_States            [LOADCTRL,CHAIN,S2,S1,S0,DMA_REQ];
        Idle                  ^b000000;
        NormalTransfer        ^b000001;

        DMA_Idle              ^b000010;
        ControlWord           ^b110001;
        TransferCount         ^b010001;
        NextChain             ^b011001;
        StartAddress          ^b010101;
        ChainComplete       = ^b010100;


"-------------------------------------------------------------------------------------
"                  DMA State Machine State Diagram
"-------------------------------------------------------------------------------------
state_diagram DMA_States

state Idle:
            "fly-by, Normal Mode, Count Up
        if  DMA_START & !DMA_BREQUEST_ & DMA_DONE_S_ & (DMA_Mode == FlyBy_Up_Normal)
                      then NormalTransfer
        else if  "fly-by Chaining Mode, Count Up
            DMA_START & CHAIN_OVF_ & (DMA_Mode == FlyBy_Up_Chain) & MASTER_
                       then ControlWord
        else if
            DMA_START & DMA_DONE_S_ & (DMA_Mode == FlyBy_Up_Chain) & MASTER_
                      then ControlWord
        else if  "non-Burst Request
            DMA_START & !DMA_REQUEST_ & DMA_BREQUEST_ & DMA_DONE_S_ & MASTER_ "Wait Until Bus is Released
                      then NormalTransfer
        else
                      Idle;

state NormalTransfer:
        if !DMA_START                              "if User aborts
                      then Idle
        else if  DMA_START & !DMA_DONE_S_
                      then Idle
        else if  DMA_BREQUEST_ & DMA_ACK       "if No Request
                      then DMA_Idle
```

```
        else                    NormalTransfer;

state DMA_Idle:
        if !DMA_START # !DMA_DONE_S_
                        then Idle
        else if (!DMA_REQUEST_ & MASTER_) # (!DMA_BREQUEST_ & MASTER_)
                        then NormalTransfer
        else                    DMA_Idle;

state ControlWord:
        if  DMA_ACK     then TransferCount
        else                    ControlWord;

state TransferCount:
        if  DMA_ACK     then NextChain
        else                    TransferCount;

state StartAddress:
        if  DMA_ACK      then ChainComplete
        else                     StartAddress;

state NextChain:
        if  DMA_ACK     then StartAddress
        else                    NextChain;

state ChainComplete:
        if  MASTER_ & !DMA_BREQUEST      then NormalTransfer  "Stay until MASTER so don't
        else if
            MASTER_ & !DMA_REQUEST
                                         then NormalTransfer
        else                                  ChainComplete;

"-------------------------------------------------------------------------------
"                   Equations
"-------------------------------------------------------------------------------
equations

"-------------------------------------------------------------------------------
"                   DMA State Machine Clock and Reset Definitions
"-------------------------------------------------------------------------------
        LOADCTRL.CLK                BCLK;
        LOADCTRL.AP                .DONE_BPST_;
        LOADCTRL.OE             =   1;
        DMA_REQ.CLK                 BCLK;
        DMA_REQ.AP                 !DONE_BPST_;
        DMA_REQ.OE                  1;
        S0.CLK                      BCLK;
        S0.AP                   =  !DONE_BPST_;
        S0.OE                       SIMULATE;
        S1.CLK                      BCLK;
        S1.AP                   =  .DONE_BPST_;
        S1.OE                   =   1;
        S2.CLK                  =   BCLK;
        S2.AP                   =  !DONE_BPST_;
        S2.OE                   =   SIMULATE;
        CHAIN.CLK               =   BCLK;
        CHAIN.AP                   .DONE_BPST_;
        CHAIN.OE                    1;

"-------------------------------------------------------------------------------
"                   Address Generation Counter Logic
"-------------------------------------------------------------------------------
        [BADDR31..BADDR8].Clk   =   BCLK;
        [CH_AD7..CH_AD2].Clk        BCLK;
        [ADR7..ADR2].Clk            BCLK;
        [ADR7..ADR2].AP            !BPST_;
        [ADR7..ADR2].OE         =   SIMULATE;
        [CH_AD7..CH_AD2].OE     =   SIMULATE;

        BADDR0                  =  !DMA_DONE_S_ & !DMA_STAT_PD_;" & MASTER_;

        BADDR1                      DMA_INTPQ & !DMA_STAT_PD_;


        [BADDR31..BADDR8].OE    =   !DMA_CHAIN_PD_;

        [BADDR7..BADDR0].OE         ADDR_OE
                                #  !DMA_CHAIN_PD_
                                #  !DMA_SOURCE_PD_
                                #  !DMA_STAT_PD_;

        [BADDR7..BADDR2]        =   Cur_Addr_B0 & !LOADCTRL & DMA_CHAIN_RD_
                                #   Chain_Addr_B0 & LOADCTRL
                                #   Chain_Addr_B0 & !DMA_CHAIN_PD_;
```

```
        CARRY_.CLK                BCLK;
        CARRY_.OE                 1;
       !CARRY_                  :=  CARRY_ & (Cur_Addr_B0 == ^h3F) & !DMA_ADDR_ENA_;

       Cur_Addr_Next             :=  Cur_Addr & !MASTER_ & !S2
                                  #  Cur_Addr & !MASTER_ & !DMA_REQ
                                  #  Addr_In  &  MASTER_ & !DMA_CHAIN_WR_
                                  #  Addr_In  & !MASTER_ &  S2 & DMA_REQ "Next Chain State
                                  #  Cur_Addr &  MASTER_ &  DMA_CHAIN_WR_;

       Cur_Addr_Next_B0          := (Cur_Addr_B0 +1) & DMA_ADDR_LOAD_ & !DMA_ADDR_ENA_ "& DMA_ACK
                                  # Cur_Addr_B0 & DMA_ADDR_LOAD_ &  DMA_ADDR_ENA_"& !DMA_ACK
                                  # Addr_In_B0 & !DMA_ADDR_LOAD_ &  DMA_ADDR_ENA_"Loading Next Chain
Pointer (Slave)
                                  # Addr_In_B0 & !DMA_ADDR_LOAD_ & !DMA_ADDR_ENA_"Loading Address During
Chaining
                                  # Cur_Addr_B0 & DMA_ADDR_LOAD_ &  DMA_ADDR_ENA_;


       Chain_Addr_Next_B0        :=  Chain_Addr_B0 & !MASTER_ & !S2
                                  #  Chain_Addr_B0 & !MASTER_ & !DMA_REQ
                                  #  Addr_In_B0 &  MASTER_ & !DMA_CHAIN_WR_
                                  #  Addr_In_B0 & !MASTER_ &  S2 &  DMA_REQ "Next Chain State
                                  #  Chain_Addr_B0 &  MASTER_ &  DMA_CHAIN_WR_;


"-------------------------------------------------------------------------------------
"                 DMA Control/Status Register Logic
"-------------------------------------------------------------------------------------
equations

        !DONE_BRST_      = !BRST_
                         # !DMA_DONE_S_ & (DMA_Mode != FlyBy_Up_Chain)
                         # !DMA_DONE_S_ & (DMA_Mode == FlyBy_Up_Chain) & !CHAIN_OVF_;

       !DMA_ADDR_OE_    = !DMA_SOURCE_RD_ & MASTER_
                        #  ADDR_OE & !LOADCTRL;

        DMA_ADDR_LOAD_.OE     1;
       !DMA_ADDR_LOAD_        !DMA_SOURCE_WR_
                         #  LOADCTRL & ADDR_PHASE "Load Chain Address
                         #  DMA_REQ & S1 & DMA_ACK; "Start Address State

       DMA_ADDR_CLK      !PCICLKIN;

       DMA_ADDR_ENA_.CLK   BCLK;
       DMA_ADDR_ENA_.OE    1;
      !DMA_ADDR_ENA_     := DMA_ACK & (DMA_States != ChainComplete) & (DMA_States != StartAddress);

       DMA_INTRQ.CLK     BCLK;
       DMA_INTRQ.AR      !BRST_;
       DMA_INTRQ.OE      1;
       DMA_INTRQ     := !INT_MODE1 &  INT_MODE0 & !DMA_DONE_S_ & !DMA_MODE1"Int on T.C. Non-Chaining
                      # !INT_MODE1 &  INT_MODE0 & !DMA_DONE_S_ &  DMA_MODE1"Int on T.C. Chaining
                      #  INT_MODE1 &  INT_MODE0 & !DMA_DONE_S_ & !CHAIN_OVF_ &  DMA_MODE1 "Int on T.C.
Chaining
                      #  DMA_INTRQ & CLS_WRITE_; "Hold until INT Clear Strobe

       DMA_COUNT_LOAD_.CLK = !BCLK;
       DMA_COUNT_LOAD_.OE  = 1;
      !DMA_COUNT_LOAD_     := !DMA_COUNT_WR_
                            #  Chain_Mode & DMA_ACK & (DMA_States == TransferCount);

      !DMA_COUNT_ENA_       DMA_ACK & !CHAIN & DMA_DONE_S_;

       DMA_DONE_S_.CLK =  BCLK;
       DMA_DONE_S_.OE  =  1;
       DMA_DONE_S_.ASET   !BRST_;
      !DMA_DONE_S_     := !DMA_DONE_
                        # !DMA_DONE_S_ &  DMA_START &  DMA_COUNT_LOAD_;

       DMA_DIR           DMA_S0;

       !CHAIN_OVF0_    !BADDR31 & !BADDR30 & !BADDR29 & !BADDR28 & !BADDR25 & !BADDR24;
       !CHAIN_OVF1_    !BADDR23 & !BADDR22 & !BADDR21 & !BADDR19 & !BADDR18 & !BADDR16;
       !CHAIN_OVF2_    !BADDR15 & !BADDR14 & !BADDR13 & !BADDR11 & !BADDR10 & !BADDR9;
       !CHAIN_OVF3_     !BADDR20 & !BADDR26;
       !CHAIN_OVF4_    !BADDR27 & !BADDR17 & !BADDR12;

       !CHAIN_OVF_ := !CHAIN_OVF4_ & !CHAIN_OVF3_ & !CHAIN_OVF2_
                    & !CHAIN_OVF1_ & !CHAIN_OVF0_;
        CHAIN_OVF_.CLK     BCLK;
        CHAIN_OVF_.OE      1;
```

```
        TC_STAGE1_S_.CLK  =  BCLK;
        TC_STAGE1_S_.OE   =  1;
        !TC_STAGE1_S_     :=  !TC_STAGE1_

end DMAC
```

5.3.3. DECODE PLD

```
module DECODE flag'-r3'
title 'Slave Decoder, PCI Status/Command Register and PCI Latency Timer
Sheet 1 of 1, Schematic Diagrams
"-------------------------------------------------------------------------------
"-------------------------------------------------------------------------------
"  This PLD implements the following functionality on the PCI Parallel Interface
"  Board:
"
"    Decodes the PCI I/O board IO Address Map:
"    The PCI I/O board utilizes an IO Address Space of
"    128 bytes.  The DECODE PLD decodes these 128 IO addresses into 32 longword
"    Read/Write IO Registers.  It does this by monitoring the least significant
"    seven address bits of the Local Address Bus (LA7-LA0) and the LDEVSEL_
"    signal from the CONFIG PLD.  The LDEVSEL_ signal is asserted active low
"    when the 24 most significant bits of the incomming IO Address are equal to
"    the value stored in the 24 most significant bits of Base Address Register 0
"    in the Primary PCI Configuration Region.  A valid address decode results in
"    the generation of a read or write control strobe used on the board to control
"    the timing of the data transfer operation.  The layout of the Parallel
"    Interface Board IO Address Map is illustrated as follows:
"
"      31                                              0 LA7   LA0 (Byte Address)
"       ------------------------------------------------
"       | Parallel Interface Control/Status Register    |     0x00
"       ------------------------------------------------
"       | Parallel Interface Interrupt Vector Register  |     0x04
"       |             (Not Implemented)                 |
"       ------------------------------------------------
"       |               Not Decoded                     |     0x08
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Output Reg. |     0x10
"       |  Byte 0 Output FIFO Read In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Output Reg. |     0x14
"       |  Byte 1 Output FIFO Read In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Output Reg. |     0x18
"       |  Byte 2 Output FIFO Read In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Output Reg. |     0x1C
"       |  Byte 3 Output FIFO Read In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Input Reg.  |     0x20
"       |  Byte 0 Input FIFO Write In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Input Reg.  |     0x24
"       |  Byte 1 Input FIFO Write In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Input Reg.  |     0x28
"       |  Byte 2 Input FIFO Write In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface Data Loopback Input Reg.  |     0x2C
"       |  Byte 3 Input FIFO Write In Diagnostic Mode   |
"       ------------------------------------------------
"       |  Parallel Interface FIFO Byte 0 Status Reg.   |     0x30
"       ------------------------------------------------
"       |  Parallel Interface FIFO Byte 1 Status Reg.   |    0x34
"       ------------------------------------------------
"       |  Parallel Interface FIFO Byte 2 Status Reg.   |     0x38
"       ------------------------------------------------
"       |  Parallel Interface FIFO Byte 3 Status Reg.   |     0x3C
"       ------------------------------------------------
"       |  DMA Controller Source Address Register       |     0x40
"       ------------------------------------------------
"       |  DMA Controller Transfer Count Register       |     0x44
"       ------------------------------------------------
"       |  DMA Controller Control/Status Register       |     0x48
"       ------------------------------------------------
"       |  DMA Controller Next Chain Pointer Register   |     0x4C
"       ------------------------------------------------
"       |  Input FIFO Data Read Register (Read Only)    |     0x50
"       ------------------------------------------------
"       |  Input FIFO Clear Command (Write Only)        |     0x54
"       ------------------------------------------------
"       |  Input FIFO AE/AF Flag Register (Write Only)  |     0x58
"       ------------------------------------------------
"       |  Not Decoded (Will Return All Ones On A Read) |     0x5C
"       ------------------------------------------------
"       |  Output FIFO Data Read Register (Read Only)   |     0x60
```

```
"      ------------------------------------------------
"      |  Output FIFO Clear Command (Write Only)      |     0xf4
"      ------------------------------------------------
"      |  Output FIFO AE/AF Flag Register (Write Only) |     0xf8
"      ------------------------------------------------
"
"   Implements a 16-Bit Shadow version of the PCI Configuration Regions's
"   Command Register at Configuation Region Offset 0x04.  This shadowed
"   version of the PCI Command Register is read/write and is used to store
"   the active PCI Configuration Region bits for real time presentation to
"   logic on the PCI I/O Board.  This 16-Bit write only
"   register is internal only with no external connections except to local
"   data bus bits BAD0   BAD15.  External Command Register connections which
"   are supported by this board are implemented by the following output
"   signals:
"
"      IOACCS_ENABLE   This signal is the active high PCI Command Register's
"                      IO Access Enable Bit.  This signal is connected to the
"                      CONFIG PLD and is used to enable PCI IO Access Decoding.
"
"      MASTER_ENABLE   This signal is the active high PCI Command Register's
"                      Master Access Enable Bit.  This signal is connected to the
"                      PCICTRL PLD and is used to enable PCI Master Accesses by
"                      the DMA Controller.  When this signal is low, then the
"                      PCI Parallel Interface Board is prohibited from
"                      initiating Master Accesses on the PCI Interface.
"
"      MEMWIV_ENABLE   This signal is the active high PCI Command Register's
"                      Memory Write And Invalidate Access Enable Bit.  This signal
"                      is connected to the PCICTRL PLD and is used to enable PCI
"                      Memory Write And Invalidate Master Access types by the
"                      DMA Controller.  When this signal is low, then the
"                      PCI Parallel Interface Board is prohibited from initiating
"                      Memory Write And Invalidate Master Accesses on the PCI Interface.
"                      If the DMA Controller is programmed to generate Memory Write
"                      And Invalidate Type Master Accesses and this bit is low, then
"                      the DMA Controller will initiate Memory Write type PCI Master
"                      Accesses instead.  The PCICTRL PLD will still use its shadow
"                      version of the Cache Line Size Register as the Burst Count size
"                      for any PCI Master transactions which have been programmed as
"                      Memory Write And Invalidate type in the DMA Control/Status
"                      Register.
"
"      PARRER_ENABLE - This signal is the active high PCI Command Register's
"                      Parity Checking and Reporting Enable Bit.  This signal
"                      is connected to the PCICTRL PLD and is used to enable PCI
"                      Parity Error Response via the SERR# signal. When this signal
"                      is low, then the PCI I/O board is prohibited
"                      from generating a System Error when a Parity Error is detected.
"                      The PCICTRL PLD is still responsible to check and generate parity
"                      regardless of the setting of this bit.  The System Error Signal
"                      Enable bit must also be set to a 1 in bit 6 of the PCI Command
"                      before the PCICTRL PLD is allow to assert or drive the PCI SERR#
"                      signal.
"
"      SYSERR_ENABLE   This signal is the active high PCI Command Register's
"                      System Error Signal Enable Bit.  This signal is connected to the
"                      PCICTRL PLD and is used to enable the System Error signal driver.
"                      When this signal is low, then the PCI I/O board
"                      is prohibited from driving the System Error signal, SERR#.
"                      This bit (and bit 6) must be set to a 1 in the PCI Configuration
"                      Region's Command Register to enable the PCI Parallel Interface
"                      board to report address or data parity errors via the PCI SERR#
"                      signal via the PCICTRL PLD.
"
"   Implements a 8-Bit Shadow version of the PCI Configuration Regions's
"   Status Register at Configuation Region Offset 0x04.  This shadowed
"   version of the PCI Command Register is read/write and is used to store
"   the active PCI Configuration Region Status bits from real time presentation
"   of status signals from the PCICTRL PLD.  This 8-Bit Status Register
"   loads real time status information into the shadowed version of this register
"   during PCI Configuration Space Read Cycles decoded at address offset 0x04.
"   The PCICTRL is prohibited from allowing any change in the real time status
"   inputs during these read operations which are indicated by an active high
"   STATUS_STROBE.   This shadowed Status Register is internal only with no external
"   connections except to local data bus bits BAD24   BAD31.  Real Time Status signal
"   connections from the PCICTRL PLD which are supported by this board are as follows:
"
"            MPARER_RPT      PCI Bus Master Parity Error Reported
"            STRG_ABORT      PCI Target Abort Issued
"            MTRG_ABORT      PCI Target Abort Received
"            MASTER_ABORT    PCI Master Abort Issued
"            APARER          Any Kind of PCI Parity Error Detected
```

```
"              LSERR          PCI System Error Issued by This Master
"
"    Once latched into the shadowed version of the PCI Configuration Region Status
"    Register, an active bit will remain active until the host writes a one into
"    the corresponding data bit location during a PCI Configuration Space Write
"    operation.
"
"    Generates the IO_CTL_WR_ signal active low in response to an IO Write
"    operation at address offset 0x00-0x1F that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"
"    Generates the IO_CTL_RD_ signal active low in response to an IO Read
"    operation at address offset 0x00-0x1F that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"
"    Generates the DMA_REG0_RD_ signal active low in response to an IO Read
"    operation at address offset 0x20 that is decoded equivalent to the address
"    stored in the PCI Configuration Regions's Base Address Register 0.  This
"    signal is used to perform a read operation of the DMA Controller's Source
"    Address Register.
"
"    Generates the DMA_REG0_WR_ signal active low in response to an IO Write
"    operation at address offset 0x20 that is decoded equivalent to the address
"    stored in the PCI Configuration Region's Base Address Register 0.  This
"    signal is used to perform a write operation of the DMA Controller's Source
"    Address Register.
"
"    Generates the DMA_REG1_RD_ signal active low in response to an IO Read
"    operation at address offset 0x24 that is decoded equivalent to the address
"    stored in the PCI Configuration Regions's Base Address Register 0.  This
"    signal is used to perform a read operation of the DMA Controller's Transfer
"    Count Register.
"
"    Generates the DMA_REG1_WR_ signal active low in response to an IO Write
"    operation at address offset 0x24 that is decoded equivalent to the address
"    stored in the PCI Configuration Region's Base Address Register 0.  This
"    signal is used to perform a write operation of the DMA Controller's Transfer
"    Count Register.
"
"    Generates the DMA_REG2_RD_ signal active low in response to an IO Read
"    operation at address offset 0x28 that is decoded equivalent to the address
"    stored in the PCI Configuration Regions's Base Address Register 0.  This
"    signal is used to perform a read operation of the DMA Controller's
"    Control/Status Register.
"
"    Generates the DMA_REG2_WR_ signal active low in response to an IO Write
"    operation at address offset 0x28 that is decoded equivalent to the address
"    stored in the PCI Configuration Region's Base Address Register 0.  This
"    signal is used to perform a write operation of the DMA Controller's
"    Control/Status Register.
"
"    Generates the DMA_REG3_RD_ signal active low in response to an IO Read
"    operation at address offset 0x2C that is decoded equivalent to the address
"    stored in the PCI Configuration Regions's Base Address Register 0.  This
"    signal is used to perform a read operation of the DMA Controller's Next
"    Pointer (Chaining Address) Register.
"
"    Generates the DMA_REG3_WR_ signal active low in response to an IO Write
"    operation at address offset 0x2C that is decoded equivalent to the address
"    stored in the PCI Configuration Region's Base Address Register 0.  This
"    signal is used to perform a write operation of the DMA Controller's Next
"    Pointer (Chaining Address) Register.
"
"    Generates the SLAVE_FIFO_RD_ signal active low in response to any 32-Bit IO
"    Read operation at address offset 0x30 that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is also generated when the DMA Controller performs a DMA Fly-By
"    IO Read/Memory Write Operation over the PCI Bus. This signal is used to read
"    data from Byte 0 of the Interface 4K byte Input FIFO Buffer.
"
"    Generates the IFIFO_CLEAR_ signal active low in response to an IO Write
"    Operation at address offset 0x34 that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is used to perform a data clear operation on the I/O Interface
"    4K byte Input FIFO Buffer with the data value written a don't care.
"
"    Generates the IFIFO_DBF signal active high in response to an IO Write
"    Operation at address offset 0x38 that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is used to program the 8-bit Almost Full/Almost Empty Flag
"    in the I/O Interface 4K byte Input FIFO Buffer.
"
"    Generates the OFIFO_WR_ signal active low in response to any 32-Bit IO
"    Write operation at address offset 0x40 that is decoded equivalent to the
```

```
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is also generated when the DMA Controller performs a DMA Fly-By
"    IO Write/Memory Read Operation over the PCI Bus. This signal is used to
"    write data into Byte 0 of the I/O Interface 4K byte Output FIFO Buffer.
"

"    Generates the OFIFO_CLEAR_ signal active low in response to an IO Write
"    Operation at address offset 0x44 that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is used to perform a data clear operation on the I/O Interface
"    4K byte Output FIFO Buffer with the data value written a don't care.
"
"  - Generates the OFIFO_DAF signal active high in response to an IO Write
"    Operation at address offset 0x48 that is decoded equivalent to the
"    address stored in the PCI Configuration Region's Base Address Register 0.
"    This signal is used to program the 8-bit Almost Full/Almost Empty Flag
"    in the I/O Interface 4K byte Output FIFO Buffer.
"
"    Generates the CLS_WRITE_ signal active low in response to a PCI Configuration
"    Space Write Operation decoded at the Cache Line Size Register.  The Cache Line
"    Size Register is shadowed in the PCICTRL PLD to be used as the Burst Count Size
"    for Memory Write And Invalidate Initiator PCI Cycles.  The CLS_WRITE_ signal is
"    generated to the PCICTRL PLD so that Cache Line Size Configuration Region Write
"    Operation can be shadowed (stored) in a 8-bit Register in the PCICTRL PLD.
"
"    Generates the signal STATUS_STROBE active high during the valid PCI data phase of
"    a PCI Configuration Space read decoded at the Status Register address.  This signal
"    is used as an internal feedback term for the equations which clock and control
"    the output enable terms of the Status Register data bits implemented in the DECODE
"    PLD.  The STATUS_STROBE signal is also used by the PCI_CTRL PLD to clear latched
"    PCI Status Register Bits.
"
"  - Generates the signal REGB0_RD_ active low to the CONFIG PLD for data byte 0 read
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB0_RD_ signal allows the
"    CONFIG PLD SRAM outputs to drive local data bits BAD0   BAD7.  This will occur for
"    valid decoded PCI Configuration Space Read Operations, PCI IO Read Operations to the
"    DMA Temporary Data Register Storage Area and during Memory-To-Memory DMA Transfers
"    under the control of the DMA_RD_ read strobe from the DMAC PLD.
"
"  - Generates the signal REGB1_RD_ active low to the CONFIG PLD for data byte 1 read
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB1_RD_ signal allows the
"    CONFIG PLD SRAM outputs to drive local data bits BAD8   BAD15.  This will occur for
"    valid decoded PCI Configuration Space Read Operations, PCI IO Read Operations to the
"    DMA Temporary Data Register Storage Area and during Memory-To-Memory DMA Transfers
"    under the control of the DMA_RD_ read strobe from the DMAC PLD.
"
"    Generates the signal REGB2_RD_ active low to the CONFIG PLD for data byte 2 read
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB2_RD_ signal allows the
"    CONFIG PLD SRAM outputs to drive local data bits BAD16   BAD23.  This will occur for
"    valid decoded PCI Configuration Space Read Operations, PCI IO Read Operations to the
"    DMA Temporary Data Register Storage Area and during Memory-To-Memory DMA Transfers
"    under the control of the DMA_RD_ read strobe from the DMAC PLD.
"
"  - Generates the signal REGB3_RD_ active low to the CONFIG PLD for data byte 3 read
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB3_RD_ signal allows the
"    CONFIG PLD SRAM outputs to drive local data bits BAD25   BAD31.  This will occur for
"    valid decoded PCI Configuration Space Read Operations, PCI IO Read Operations to the
"    DMA Temporary Data Register Storage Area and during Memory-To-Memory DMA Transfers
"    under the control of the DMA_RD_ read strobe from the DMAC PLD.
"
"    Generates the signal REGB0_WR_ active low to the CONFIG PLD for data byte 0 write
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB0_WR_ signal allows the
"    CONFIG PLD SRAM data inputs to be driven by local data bits BAD0   BAD7.  This will
"    occur for valid decoded PCI Configuration Space Write Operations, PCI IO Write
"    Operations to the DMA Temporary Data Register Storage Area and during Memory-To-Memory
"    DMA Transfers under the control of the DMA_WR_ write strobe from the DMAC PLD.
"
"    Generates the signal REGB1_WR_ active low to the CONFIG PLD for data byte 1 write
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB1_WR_ signal allows the
"    CONFIG PLD SRAM data inputs to be driven local data bits BAD8   BAD15.  This will
"    occur for valid decoded PCI Configuration Space Write Operations, PCI IO Write
"    Operations to the DMA Temporary Data Register Storage Area and during Memory-To-Memory
```

```
"    DMA Transfers under the control of the DMA_WR_ write strobe from the DMAC PLD.
"

"    Generates the signal REGB2_WR_ active low to the CONFIG PLD for data byte 2 write
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB2_WR_ signal allows the
"    CONFIG PLD SRAM data inputs to drivn bye local data bits BAD16   BAD23.  This will
"    occur for valid decoded PCI Configuration Space Write Operations, PCI IO Write
"    Operations to the DMA Temporary Data Register Storage Area and during Memory-To-Memory
"    DMA Transfers under the control of the DMA_WR_ write strobe from the DMAC PLD.
"
"    Generates the signal REGB3_WR_ active low to the CONFIG PLD for data byte 3 write
"    accesses to either the PCI Configuration Region of the DMA Temporary Data Register
"    Storage Area.  Both of these memory spaces are implemented in the SRAM contained
"    within the CONFIG PLD.  The active low state of the REGB3_WR_ signal allows the
"    CONFIG PLD SRAM data inputs to be driven by local data bits BAD25   BAD31.  This will
"    occur for valid decoded PCI Configuration Space Write Operations, PCI IO Write
"    Operations to the DMA Temporary Data Register Storage Area and during Memory-To-Memory
"    DMA Transfers under the control of the DMA_WR_ write strobe from the DMAC PLD.
"
"    Generates the LT_OVF_ active low in response to the Latency Timer counting down to
"    zero.   LT_OVF_ is synchronized to LCLK and presented to the PCICTRL PLD
"    PLD to instruct it to disconnect from the PCI Bus is the PCI Arbiter has removed its
"    Grant.  This signal is designated as the Latency Timer Overflow Output.
"
"    Implements an 8-Bit PCI Latency Timer which is activated when the board arbitrates for and
"    becomes a PCI Initiator.  The Latency Timer prevents the board from holding the PCI Bus as
"    a Initiator for more than the programmed number of PCI Clocks in the Configuration Regions
"    Latency Timer field.  When this counter overflows by counting down to zero, the signal,
"    LT_OVF_, is asserted active low to the PCICTRL PLD to instruct it to release the PCI bus
"    upon completion of the current data phase provided that its PCI Bus Grant has been
"    relinquished by the Host PCI Bus Arbitrator.
"
"    Generates demultiplexed versions of the PCI Command Signals
"    CMD0, CMD1, CMD2 and CMD3.  These are used by the DECODE and CONFIG PLDs
"    to decode the type of PCI transaction being executed during a PCI Frame.

"-------------------------------------------------------------------------------------------
"                  Declarations
"-------------------------------------------------------------------------------------------

                   DECODE     device 'ifx780fp' ,

"-------------------------------------------------------------------------------------------
"                  Inputs Pins
"-------------------------------------------------------------------------------------------
        CLK66                   pin  52;  "Local (up to 66Mhz) x2 PCI Bus Clock
        BCLK                    pin   1;  "Local PCI Bus Clock for combinational logic
        ADDR_CLOCK              pin 118;  "Address Demultiplexing Clock
        BRST_                   pin   2;  "Local Active Low PCI Reset
        BIDSEL                  pin   3;  "Buffered Configuration Region Select
        BTRDY_                  pin   4;  "Local PCI Target Ready#
        BIRDY_                  pin   5;  "Local PCI Initiator Ready#
        BFRAME_                 pin  29;  "Buffered PCI FRAME# signal
        LFRAME_                 pin  33;  "Local PCI FRAME# signal
        MASTER_                 pin  34;  "Local PCI Device is PCI Bus Master#
        DMA_DONE_               pin  35;  "DMA Transfer Count Complete#
        DMA_DONE_S_             pin  22;  "DMA Transfer Count Complete Latched#
        DMA_ACK                 pin  36;  "DMA Cycle Acknowledge
        DMA_DIRECTION           pin  37;  "DMA Cycle Direction
        CHAIN                   pin  38;  "DMA Chaining Parameter Fetching Operation
        LOADCTRL                pin  67;  "DMA State Machine Chain Load DMA Control Register
        DATA_PHASE_             pin  68;  "Valid PCI Data Phase Detected
        CMD_ENABLE_             pin  25;  "Valid PCI Command Enable#

        BC_BE0_                 pin  69;  "Local PCI Byte Enable 0#
        BC_BE1_                 pin  70;  "Local PCI Byte Enable 1#
        BC_BE2_                 pin  71;  "Local PCI Byte Enable 2#
        BC_BE3_                 pin  99;  "Local PCI Byte Enable 3#

        CMD0                    pin  55;  "Local PCI Command Bit 0
        CMD1                    pin  10;  "Local PCI Command Bit 1
        CMD2                    pin  12;  "Local PCI Command Bit 2
        CMD3                    pin  88;  "Local PCI Command Bit 3

        LDEVSEL_                pin 100;  "Local PCI Device Select#
```

```
"-------------------------------------------------------------------------------------------------
"        PCI Local Address Bits To Decode
"-------------------------------------------------------------------------------------------------

       LA7                    pin  48;  "Local Address Bit 7
       LA6                    pin  46;  "Local Address Bit 6
       LA5                    pin  43;  "Local Address Bit 5
       LA4                    pin  45;  "Local Address Bit 4
       LA3                    pin  47;  "Local Address Bit 3
       LA2                    pin  42;  "Local Address Bit 2

"-------------------------------------------------------------------------------------------------
"        Latched Latency Timer Register Values
"-------------------------------------------------------------------------------------------------

        LTC7                  node 40;  "Latched Latency Timer Value 7
        LTC6                  node 39;  "Latched Latency Timer Value 6
        LTC5                  node 41;  "Latched Latency Timer Value 5
        LTC4                  node 30;  "Latched Latency Timer Value 4
        LTC3                  pin  30;  "Latched Latency Timer Value 3
        LTC2                  pin  31;  "Latched Latency Timer Value 2
        LTC1                  pin 101;  "Latched Latency Timer Value 1
        LTC0                  pin 102;  "Latched Latency Timer Value 0
        LT_WRITE_             pin 124;  "Latency Timer Write Strobe#

"-------------------------------------------------------------------------------------------------
"        PCI Status Register Inputs From PCI Master/Slave State Machines
"-------------------------------------------------------------------------------------------------

       MPARER_RPT             pin 103;  "PCI Bus Master Parity Error Reported
       STRG_ABORT             pin 104;  "PCI Target Abort Issued
       MTRG_ABORT             pin  35;  "PCI Target Abort Received
       MASTER_ABORT           pin  40;  "PCI Master Abort Received
       APARER                 pin  41;  "Any Kind of PCI Parity Error Detected
       LSERR                  pin  32;  "PCI System Error Issued by This Master

"-------------------------------------------------------------------------------------------------
"        PCI Valid Command Register Outputs To PCI Master/Slave State Machines
"-------------------------------------------------------------------------------------------------

       IOACCS_ENABLE          pin  89;  "PCI Command Register IO Access Enable
       MASTER_ENABLE          pin  49;  "PCI Command Register Master Enable
       MEMWIV_ENABLE          pin   6;  "PCI Command Register Memory Write and Invalidate
       PAPERR_ENABLE          pin   7;  "PCI Command Register Parity Error Enable
       SYSERR_ENABLE          pin   8;  "PCI Command Register System Error Enable

"-------------------------------------------------------------------------------------------------
"                  PCI Status Register Bi-Directional Pins
"-------------------------------------------------------------------------------------------------

       STATUS_B8              pin  98;  "Status Register Data Bit 8
       STATUS_B9              pin  97;  "Status Register Data Bit 9
       STATUS_B10             pin  96;  "Status Register Data Bit 10
       STATUS_B11             pin  95;  "Status Register Data Bit 11
       STATUS_B12             pin  94;  "Status Register Data Bit 12
       STATUS_B13             pin  92;  "Status Register Data Bit 13
       STATUS_B14             pin  91;  "Status Register Data Bit 14
       STATUS_B15             pin  90;  "Status Register Data Bit 15

       BAD24                  pin  98;  "Buffered Data/Address Bit 24
       BAD25                  pin  97;  "Buffered Data/Address Bit 25
       BAD26                  pin  96;  "Buffered Data/Address Bit 26
       BAD27                  pin  95;  "Buffered Data/Address Bit 27
       BAD28                  pin  94;  "Buffered Data/Address Bit 28
       BAD29                  pin  92;  "Buffered Data/Address Bit 29
       BAD30                  pin  91;  "Buffered Data/Address Bit 30
       BAD31                  pin  90;  "Buffered Data/Address Bit 31

       COMMAND_B0             pin  82;  "Command Register Data Bit 0
       COMMAND_B1             pin  81;  "Command Register Data Bit 1
       COMMAND_B2             pin  80;  "Command Register Data Bit 2
       COMMAND_B3             pin  79;  "Command Register Data Bit 3
       COMMAND_B4             pin  78;  "Command Register Data Bit 4
       COMMAND_B5             pin  76;  "Command Register Data Bit 5
       COMMAND_B6             pin  75;  "Command Register Data Bit 6
       COMMAND_B7             pin  74;  "Command Register Data Bit 7

       COMMAND_B8             pin  64;  "Command Register Data Bit 8
       COMMAND_B9             pin  63;  "Command Register Data Bit 9
       COMMAND_B10            pin  62;  "Command Register Data Bit 10
       COMMAND_B11            pin  61;  "Command Register Data Bit 11
       COMMAND_B12            pin  60;  "Command Register Data Bit 12
       COMMAND_B13            pin  58;  "Command Register Data Bit 13
       COMMAND_B14            pin  57;  "Command Register Data Bit 14
```

```
        COMMAND_B15             pin  56;  "Command Register Data Bit 15

        BAD0                    pin  82;  "Status Register Data Bit 0
        BAD1                    pin  81;  "Status Register Data Bit 1
        BAD2                    pin  80;  "Status Register Data Bit 2
        BAD3                    pin  79;  "Status Register Data Bit 3
        BAD4                    pin  78;  "Status Register Data Bit 4
        BAD5                    pin  76;  "Status Register Data Bit 5
        BAD6                    pin  75;  "Status Register Data Bit 6
        BAD7                    pin  74;  "Status Register Data Bit 7

        BAD8                    pin  64;  "Status Register Data Bit 8
        BAD9                    pin  63;  "Status Register Data Bit 9
        BAD10                   pin  62;  "Status Register Data Bit 10
        BAD11                   pin  61;  "Status Register Data Bit 11
        BAD12                   pin  60;  "Status Register Data Bit 12
        BAD13                   pin  58;  "Status Register Data Bit 13
        BAD14                   pin  57;  "Status Register Data Bit 14
        BAD15                   pin  56;  "Status Register Data Bit 15

"-------------------------------------------------------------------------------------------
"                 Outputs and Feedback Terms
"-------------------------------------------------------------------------------------------

        IO_CTL_WR_              pin 106;  "I/O Control Register Write#
        IO_STA_RD_              pin 105;  "I/O Status Register Read#
        SLAVE_FIFO_RD_          pin  72;  "Input FIFO 32-Bit Read Strobe in Slave Mode#
        IFIFO_CLEAR_            pin 107;  "Input FIFO Clear Strobe#
        IFIFO_DBF               pin 109;  "Input FIFO Define Flag Strobe
        OFIFO_WR_               pin 122;  "Output FIFO 32-Bit Write Strobe#
        OFIFO_CLEAR_            pin 113;  "Output FIFO Clear Strobe#
        OFIFO_DAF               pin 126;  "Output FIFO Define Flag Strobe
        FIFO_GBA                pin   9;  "FIFO PCI Side Transceiver Control


        DMA_REG0_RD_            pin 108;  "DMA Controller Source Address Register Read#
        DMA_REG1_RD_            pin 112;  "DMA Controller Transfer Count Register Read#
        DMA_REG2_RD_            pin 111;  "DMA Controller Control/Status Register Read#
        DMA_REG3_RD_            pin 127;  "DMA Controller Next Pointer Register Read#
        DMA_REG0_WR_            pin 128;  "DMA Controller Source Address Register Write#
        DMA_REG1_WR_            pin 129;  "DMA Controller Transfer Count Register Write#
        DMA_REG2_WR_            pin 123;  "DMA Controller Control/Status Register Write#
        DMA_REG3_WR_            pin 130;  "DMA Controller Next Pointer Register Write#

        CLS_WRITE_              pin  73;  "Cache Line Size Configuration Region Write Strobe#
        STATUS_STROBE           pin  54;  "PCI Configuration Region Status Register Strobe#

        REGB0_RD_               pin 120;  "Config Region and DMA Temp Data Storage Byte 0 Read Strobe#
        REGB1_RD_               pin 121;  "Config Region and DMA Temp Data Storage Byte 1 Read Strobe#
        REGB2_RD_               pin 115;  "Config Region and DMA Temp Data Storage Byte 2 Read Strobe#
        REGB3_RD_               pin 114;  "Config Region and DMA Temp Data Storage Byte 3 Read Strobe#

        REGB0_WR_               pin  16;  "Config Region and DMA Temp Data Storage Byte 0 Write Strobe#
        REGB1_WR_               pin  15;  "Config Region and DMA Temp Data Storage Byte 1 Write Strobe#
        REGB2_WR_               pin  14;  "Config Region and DMA Temp Data Storage Byte 2 Write Strobe#
        REGB3_WR_               pin  13;  "Config Region and DMA Temp Data Storage Byte 3 Write Strobe#

"-------------------------------------------------------------------------------------------
"                 PCI Latency Timer Pin Definitions
"-------------------------------------------------------------------------------------------

        LT_OVF                  pin  28;  "Latency Timer Overflow Output

        LTCD7                  node  24;  "Latency Timer Data Bit 7
        LTCD6                  node  22;  "Latency Timer Data Bit 6
        LTCD5                  node  23;  "Latency Timer Data Bit 5
        LTCD4                  node  32;  "Latency Timer Data Bit 4
        LTCD3                  node  29;  "Latency Timer Data Bit 3
        LTCD2                  node  31;  "Latency Timer Data Bit 2
        LTCD1                  node  26;  "Latency Timer Data Bit 1
        LTCD0                  node  25;  "Latency Timer Data Bit 0

        LTimer     [LTCD7,LTCD6,LTCD5,LTCD4,LTCD3,LTCD2,LTCD1,LTCD0];
        LTimerHi   [LTCD7,LTCD6,LTCD5,LTCD4];
        LTimerLo   [LTCD3,LTCD2,LTCD1,LTCD0];
        LTimerIn   [LTCD7.fb,LTCD6.fb,LTCD5.fb,LTCD4.fb,
                    LTCD3.fb,LTCD2.fb,LTCD1.fb,LTCD0.fb];
        LTREG_IN        [BAD15,BAD14,BAD13,BAD12,BAD11,LTC2,LTC1,LTC0];
        LTREG_INHI      [BAD15,BAD14,BAD13,BAD12];
        LTREG_OUT       [LTC7.fb,LTC6.fb,LTC5.fb,LTC4.fb,LTC3,LTC2,LTC1,LTC0];
        LTREG_OUT_SIM   [LTC7,LTC6,LTC5,LTC4];
        LTREG_OUT_HI    [LTC7,LTC6,LTC5,LTC4];
        LTREG_OUT_LO    [LTC3,LTC2,LTC1,LTC0];
        LTREG_INLO      [LTC3,LTC2,LTC1,LTC0];
```

```
"-------------------------------------------------------------------------------
"                  Set Definitions and Constants
"-------------------------------------------------------------------------------

        H,L,X,C,Z,SIMULATE   1,0,.X.,.C.,.Z.,1;
"-------------------------------------------------------------------------------
"       PCI Command Definitions
"-------------------------------------------------------------------------------
        PCI_CMD_BE              [BC_BE3_,BC_BE2_,BC_BE1_,BC_BE0_];
        PCI_Command             [CMD3,CMD2,CMD1,CMD0];
        InterruptAcknowledge  = [  0 ,  0 ,  0 ,  0 ];
        SpecialCycle          = [  0 ,  0 ,  0 ,  1 ];
        IO_Read                 [  0 ,  0 ,  1 ,  0 ];
        IO_Write                [  0 ,  0 ,  1 ,  1 ];
        Reserved1               [  0 ,  1 ,  0 ,  0 ];
        Reserved2               [  0 ,  1 ,  0 ,  1 ];
        MemoryRead              [  0 ,  1 ,  1 ,  0 ];
        MemoryWrite             [  0 ,  1 ,  1 ,  1 ];
        Reserved3               [  1 ,  0 ,  0 ,  0 ];
        Reserved4               [  1 ,  0 ,  0 ,  1 ];
        ConfigurationRead       [  1 ,  0 ,  1 ,  0 ];
        ConfigurationWrite      [  1 ,  0 ,  1 ,  1 ];
        MemoryReadMultiple      [  1 ,  1 ,  0 ,  0 ];
        DualAddressCycle      = [  1 ,  1 ,  0 ,  1 ];
        MemoryReadLine          [  1 ,  1 ,  1 ,  0 ];
        MemoryWriteInvalidate   [  1 ,  1 ,  1 ,  1 ];

        LocalData    =  [BAD7,BAD6,BAD5,BAD4,BAD3,BAD2,BAD1,BAD0];
        TimerData    =  [BAD15,BAD14,BAD13,BAD12,BAD11,BAD10,BAD9,BAD8];
        LocalAddress    [LA6,LA5,LA4,LA3,LA2]; "0x00   0xFF
        LocalSpace      [LA7,LA6];

        IOControlStatus            ^h0;  "0x00   0x3F
        DMASourceAddressReg        ^h10; "0x40   0x43
        DMATransferCountReg        ^h11; "0x44   0x47
        DMAControlStatusReg        ^h12; "0x48   0x4B
        DMANextPointerReg          ^h13; "0x4C   0x4F
        InputFIFO                =  ^h14; "0x50   0x53
        ClearInputFIFO           =  ^h15; "0x54   0x57
        WriteInputFIFOFlag       =  ^h16; "0x58   0x5B
        OutputFIFO               =  ^h18; "0x60   0x63
        ClearOutputFIFO          =  ^h19; "0x64   0x67
        WriteOutputFIFOFlag        ^h1A; "0x68   0x6B
        InterruptClear             ^h1B; "0x6C - 0x6F

        DMATempSpace               [1,X,X,X,X,X,X]; "0x80   0xFF
        ConfigRegionSpace          [0,X,X,X,X,X,X]; "0x00   0x7F

        ConfigAddress      [ LA6 ,LA5 ,LA4 ,LA3 ,LA2 ,BAD1 ,BAD0 ];
        LatencyTimer       [  0 ,  0 ,  0 ,  1 ,  1 ,  X  ,  X  ];
        StatusRegister     [  0 ,  0 ,  0 ,  0 ,  1 ,  X  ,  X  ];
        CommandRegister    [  0 ,  0 ,  0 ,  0 ,  1 ,  X  ,  X  ];
        CacheLineSize      [  0 ,  0 ,  0 ,  1 ,  1 ,  X  ,  X  ];
        AddressZero        [  0 ,  0 ,  0 ,  0 ,  0 ,  X  ,  X  ];
        BaseAddress1       [  0 ,  0 ,  1 ,  0 ,  1 ,  X  ,  X  ];
        BaseAddress2       [  0 ,  0 ,  1 ,  1 ,  0 ,  X  ,  X  ];
        BaseAddress1_3     [  0 ,  0 ,  1 ,  X ,  1 ,  X  ,  X  ];
        BaseAddress3       [  0 ,  0 ,  1 ,  1 ,  1 ,  X  ,  X  ];
        BaseAddress4       [  0 ,  1 ,  X ,  X ,  0 ,  X  ,  X  ];
        BaseAddress5       [  0 ,  1 ,  X ,  0 ,  1 ,  X  ,  X  ];
        BaseAddress4_5     [  0 ,  1 ,  X ,  X ,  X ,  X  ,  X  ];
        Reserved2C         [  0 ,  1 ,  0 ,  1 ,  1 ,  X  ,  X  ];


"-------------------------------------------------------------------------------
"                  Equations
"-------------------------------------------------------------------------------
equations

        !IO_CTL_WR_         !LFRAME_ & (LocalSpace == IOControlStatus)
                             & !DATA_PHASE_ & !LDEVSEL_
                             & (PCI_Command == IO_Write) & BRST_
                          # !LFRAME_ & (LocalAddress == WriteInputFIFOFlag)
                             & !LDEVSEL_
                             & (PCI_Command == IO_Write) & BRST_;

        !IO_STA_RD_         !LFRAME_ & (LocalSpace == IOControlStatus)
                          & !LDEVSEL_
                          & (PCI_Command == IO_Read) & BRST_;

"Equations for Decoding and Generating The DMA Source Address Register
"Read And Write Strobes:
```

```
          !DMA_REG0_RD_      !LFRAME_ & (LocalAddress == DMASourceAddressReg)
                           & !LDEVSEL_
                           & (PCI_Command == IO_Read) & BRST_;

          !DMA_REG0_WR_      !LFRAME_ & (LocalAddress == DMASourceAddressReg)
                           & !DATA_PHASE_ & !LDEVSEL_
                           & (PCI_Command == IO_Write) & BRST_;

"Equations for Decoding and Generating The DMA Transfer Count Register
"Read And Write Strobes:

          !DMA_REG1_RD_      !LFRAME_ & (LocalAddress == DMATransferCountReg)
                           & !LDEVSEL_
                           & (PCI_Command == IO_Read) & BRST_;

          !DMA_REG1_WR_      !LFRAME_ & (LocalAddress == DMATransferCountReg)
                           & !DATA_PHASE_ & !LDEVSEL_
                           & (PCI_Command == IO_Write) & BRST_;

"Equations for Decoding and Generating The DMA Control/Status Register
"Read And Write Strobes:

          !DMA_REG2_RD_      !LFRAME_ & (LocalAddress == DMAControlStatusReg)
                           & !LDEVSEL_
                           & (PCI_Command == IO_Read) & BRST_;

          !DMA_REG2_WR_      !LFRAME_ & (LocalAddress == DMAControlStatusReg)
                           & !DATA_PHASE_ & !LDEVSEL_ & !BCLK
                           & (PCI_Command == IO_Write) & BRST_
                           # !CMD_ENABLE_ & CHAIN & LOADCTRL & DMA_ACK & !BCLK;

"Equations for Decoding and Generating The DMA Next Pointer Address Pointer
"Read And Write Strobes:

          !DMA_REG3_RD_    = !LFRAME_ & (LocalAddress == DMANextPointerReg)
                           & !LDEVSEL_
                           & (PCI_Command == IO_Read) & BRST_
                           #  LDEVSEL_ &  LOADCTRL & CHAIN & DATA_PHASE_;

          !DMA_REG3_WR_      !LFRAME_ & (LocalAddress == DMANextPointerReg)
                           &  DATA_PHASE_ & !LDEVSEL_
                           & (PCI_Command == IO_Write) & BRST_;

"Equation For Decoding and Generating The Input FIFO Clear Strobe:
           IFIFO_CLEAR_.CLK   BCLK;
           IFIFO_CLEAR_.OE    1;
           !IFIFO_CLEAR_   := !LFRAME_ & (LocalAddress == ClearInputFIFO)
                              & !DATA_PHASE_ & !LDEVSEL_
                              & (PCI_Command == IO_Write) & BRST_
                            # !BRST_;
"                            # !LFRAME_ & (LocalAddress == WriteInputFIFOFlag)
"                              & !LDEVSEL_ & !BIRDY_ & !BTRDY_
"                              & (PCI_Command == IO_Write) & BRST_;

"Equation To Decode and Generate The Input FIFO
"Almost Full/Almost Emtpy Flag Write Strobe:
"          IFIFO_DBF.CLK    =  BCLK;
"          IFIFO_DBF.OE       1;

"           IFIFO_DBF           1;
          !IFIFO_DBF          !IFIFO_CLEAR_ & BRST_;

"          !IFIFO_DBF       := !LFRAME_ & (LocalAddress == WriteInputFIFOFlag)
                              "& !DATA_PHASE_ & !LDEVSEL_
                              "& (PCI_Command == IO_Write) & BRST_
                              "!IFIFO_CLEAR_ & BRST_;

"Equations For Decoding and Generating the Input FIFO Read Strobes in Slave Mode:

           SLAVE_FIFO_RD_ := !LFRAME_ & (LocalAddress == InputFIFO)
                             & !LDEVSEL_ & !DATA_PHASE_
                             & (PCI_Command == IO_Read) & BRST_;
           SLAVE_FIFO_RD_.CLK  = !CLK66;
           SLAVE_FIFO_RD_.OE   =  1;

"Equation To Decode and Generate The Output FIFO Clear Strobe:

          !OFIFO_CLEAR_      !LFRAME_ & (LocalAddress == ClearOutputFIFO)
                             & !DATA_PHASE_ & !LDEVSEL_
                             & (PCI_Command == IO_Write) & BRST_
                           # !LFRAME_ & (LocalAddress == WriteOutputFIFOFlag)
                             & !LDEVSEL_ & !BIRDY_ & !BTRDY_
                             & (PCI_Command == IO_Write) & BRST_
                           # !BRST_;
```

```
"Equation To Decode and Generate The Output FIFO
"Almost Full/Almost Emtpy Flag Write Strobe:

           OFIFO_DAF.CLK      BCLK;
           OFIFO_DAF.OE       1;
          !OFIFO_DAF       := !LFRAME_ & (LocalAddress == WriteOutputFIFOFlag)
                              & !DATA_PHASE_ & !LDEVSEL_
                              & (PCI_Command == IO_Write) & BRST_
                              # !OFIFO_CLEAR_ & BRST_;

"Equations To Decode and Generate the Output FIFO Write Strobes:

           OFIFO_WR_     := !LFRAME_ & (LocalAddress == OutputFIFO)
                              & !DATA_PHASE_ & !LDEVSEL_
                              & (PCI_Command == IO_Write) & BRST_
                            #  DMA_ACK & !CHAIN & !DMA_DIRECTION & !BCLK & BRST_ & DMA_DONE_ & DMA_DONE_S_;
           OFIFO_WR_.AP  = !DMA_DONE_;
           OFIFO_WR_.CLK   !CLK66;
           OFIFO_WR_.OE  =  1;

"Equation For enabling the 32-Bit PCI transceiver side of the FIFO Buffers

           FIFO_GBA       := !CMD_ENABLE_ & DMA_DIRECTION & !LFRAME_ & !CHAIN "Enable After Address Phase
                             # !LDEVSEL_ & (LocalAddress==InputFIFO)
                                          & (PCI_Command==IO_Read);
           FIFO_GBA.CLK      BCLK;
           FIFO_GBA.AP       CHAIN;
           FIFO_GBA.OE       1;

"Equations For Controlling Latency Timer Operation

          LTimer.CLK    BCLK;
          LTimer.ASET    !BRST_;
          LTREG_OUT_HI.ASET = !BRST_;

          LTCD7      :=  LTC7.fb & MASTER_;
          LTCD6      :=  LTC6.fb & MASTER_;
          LTCD5      :=  LTC5.fb & MASTER_;
          LTCD4      :=  LTC4.fb & MASTER_;
          LTCD3      :=  LTC3 & MASTER_;
          LTCD2      :=  LTC2 & MASTER_;
          LTCD1      :=  LTC1 & MASTER_;
          LTCD0      :=  LTC0 & MASTER_;

          LT_OVF.CLK     BCLK;
          LT_OVF     := !LTCD7 & !LTCD6 & !LTCD5 & !LTCD4
                        & !LTCD3 & !LTCD2 & !LTCD1 & !LTCD0;
          LT_OVF.OE     1;

         !LT_WRITE_       !LFRAME_ & (ConfigAddress == LatencyTimer)
                        & !DATA_PHASE_ & BRST_
                        & (PCI_Command == ConfigurationWrite) & !BC_BE1_;

"Equation to Latch Latency Timer Value Into Internal Register:

          LTC7         := BAD15 & !LT_WRITE_
                          # LTC7.fb & LT_WRITE_ & BRST_;
          LTC7.OE         1;
          LTC7.CLK        BCLK;

          LTC6         := BAD14 & !LT_WRITE_
                          # LTC6.fb & LT_WRITE_ & BRST_;
          LTC6.OE      = 1;
          LTC6.CLK     = BCLK;

          LTC5         := BAD13 & !LT_WRITE_
                          # LTC5.fb & LT_WRITE_ & BRST_;
          LTC5.OE      = 1;
          LTC5.CLK        BCLK;

          LTC4         := BAD12 & !LT_WRITE_
                          # LTC4.fb & LT_WRITE_ & BRST_;
          LTC4.OE         1;
          LTC4.CLK        BCLK;

"Equations For Accessing Configuration Region and DMA Temporary Data Storage SRAM:

          !REGB3_WR_     = !LDEVSEL_ & (PCI_Command == ConfigurationWrite)
                         &  BRST_ & !DATA_PHASE_ & !BC_BE3_ & CMD_ENABLE_
                         & (ConfigAddress != BaseAddress1_3)
                         & (ConfigAddress != BaseAddress2)
                         & (ConfigAddress != BaseAddress4_5)
                         & (ConfigAddress != AddressZero)
```

```
                    # !LFRAME_ & !BC_BE3_ & BRST_
                              & !DATA_PHASE_ & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Write) & CMD_ENABLE_;


          !REGB2_WR_        !LDEVSEL_ & (PCI_Command == ConfigurationWrite)
                            &  BRST_ & !DATA_PHASE_ & !BC_BE2_ & CMD_ENABLE_
                            & (ConfigAddress != BaseAddress1_3)
                            & (ConfigAddress != BaseAddress2)
                            & (ConfigAddress != StatusRegister)
                            & (ConfigAddress != BaseAddress4_5)
                            & (ConfigAddress != AddressZero)
                    # !LFRAME_ & !BC_BE2_ & BRST_
                              & !DATA_PHASE_ & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Write) & CMD_ENABLE_;


          !REGB1_WR_        !LDEVSEL_ & (PCI_Command == ConfigurationWrite)
                          &  BRST_ & !DATA_PHASE_ & !BC_BE1_ & CMD_ENABLE_
                          & (ConfigAddress != BaseAddress1_3)
                          & (ConfigAddress != BaseAddress4_5)
                          & (ConfigAddress != BaseAddress2)
                          & (ConfigAddress != AddressZero)
                          # !LFRAME_ & !BC_BE1_ & BRST_
                              & !DATA_PHASE_ & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Write) & CMD_ENABLE_;


          !REGB0_WR_        !LDEVSEL_ & (PCI_Command == ConfigurationWrite)
                          &  BRST_ & !DATA_PHASE_ & !BC_BE0_ & CMD_ENABLE_
                          & (ConfigAddress != BaseAddress1_3)
                          & (ConfigAddress != BaseAddress2)
                          & (ConfigAddress != Reserved2C)
                          & (ConfigAddress != BaseAddress4)
                          & (ConfigAddress != BaseAddress5)
                          & (ConfigAddress != AddressZero)
                          # !LFRAME_ & !BC_BE0_ & BRST_
                              & !DATA_PHASE_ & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Write) & CMD_ENABLE_;

          !REGB0_RD_      = !LDEVSEL_    & (PCI_Command == ConfigurationRead) & CMD_ENABLE_
                          &  (ConfigAddress != StatusRegister)
                          & BRST_ & !BC_BE0_;

          !REGB1_RD_      = !LDEVSEL_ & (PCI_Command == ConfigurationRead) & CMD_ENABLE_
                          &  (ConfigAddress != StatusRegister)
                          &  BRST_    & !BC_BE1_
                          # !LFRAME_ & BRST_
                              & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Read) & CMD_ENABLE_ & !BC_BE1_;

          !REGB2_RD_        !LDEVSEL_ & (PCI_Command == ConfigurationRead) & CMD_ENABLE_
                          &  BRST_    & !BC_BE2_
                          # !LFRAME_ & BRST_
                              & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Read) & CMD_ENABLE_ & !BC_BE2_;

          !REGB3_RD_        !LDEVSEL_    & (PCI_Command == ConfigurationRead)  & CMD_ENABLE_
                          &  (ConfigAddress != StatusRegister)
                          &  BRST_    & !BC_BE3_
                          # !LFRAME_ & BRST_
                              & !LDEVSEL_ & (LocalAddress == DMAControlStatusReg)
                              & (PCI_Command == IO_Read) & CMD_ENABLE_ & !BC_BE3_;

"Equation to Generate Cache Line Size Shadow Register Write Strobe:

          !CLS_WRITE_         !LFRAME_ & (LocalAddress == InterruptClear)
                          &  !LDEVSEL_ & BRST_
                          & (PCI_Command == IO_Write) & CMD_ENABLE_ & !BC_BE0_;

"Equations For Setting, Clearing And Enabling Command Register Bits:

          COMMAND_B15     := 0;
          COMMAND_B15.CLK   (ConfigAddress == CommandRegister)
                          & !DATA_PHASE_ & !LDEVSEL_
                          & (PCI_Command == ConfigurationWrite) & !BC_BE1_;
          COMMAND_B15.OE  = (ConfigAddress == CommandRegister)
                          & !LDEVSEL_
                          & (PCI_Command == ConfigurationRead) & !BC_BE1_;
          COMMAND_B15.AR    !BRST_;

          COMMAND_B14    := 0;
          COMMAND_B14.CLK   (ConfigAddress == CommandRegister)
                          & !DATA_PHASE_ & !LDEVSEL_
```

```
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_;
COMMAND_B14.OE  (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_.
COMMAND_B14.AR  !BRST_;

COMMAND_B13    := 0;
COMMAND_B13.CLK  (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_.
COMMAND_B13.OE  (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_.
COMMAND_B13.AR  !BRST_;

COMMAND_B12    := 0;
COMMAND_B12.CLK  (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_,
COMMAND_B12.OE  (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_;
COMMAND_B12.AR  !BRST_;

COMMAND_B11    := 0;
COMMAND_B11.CLK  (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_;
COMMAND_B11.OE  (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_,
COMMAND_B11.AR  !BRST_;

COMMAND_B10    := 0;
COMMAND_B10.CLK  (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_.
COMMAND_B10.OE = (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_,
COMMAND_B10.AR  !BRST_;

COMMAND_B9     := 0;
COMMAND_B9.CLK  = (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE1_.
COMMAND_B9.OE   (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_,
COMMAND_B9.AR   !BRST_;

COMMAND_B8     := BAD8;
COMMAND_B8.CLK   (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B8.OE   (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE1_.
COMMAND_B8.AR   !BRST_;

COMMAND_B7     := 0;
COMMAND_B7.CLK   (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE0_,
COMMAND_B7.OE   (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B7.AR   !BRST_;

COMMAND_B6     := BAD6;
COMMAND_B6.CLK   (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B6.OE  = (ConfigAddress == CommandRegister)
                & !LDEVSEL_
                & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B6.AR   !BRST_;

COMMAND_B5     := BAD5;
COMMAND_B5.CLK   (ConfigAddress == CommandRegister)
                & !DATA_PHASE_ & !LDEVSEL_
                & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B5.OE   (ConfigAddress == CommandRegister)
                & !LDEVSEL_
```

```
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B5.AR       !BRST_;

COMMAND_B4      := BAD4;
COMMAND_B4.CLK   = (ConfigAddress == CommandRegister)
                    & !DATA_PHASE_ & !LDEVSEL_
                    & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B4.OE    (ConfigAddress == CommandRegister)
                    & !LDEVSEL_
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B4.AR   = !BRST_;

COMMAND_B3      := BAD3;
COMMAND_B3.CLK   = (ConfigAddress == CommandRegister)
                    & !DATA_PHASE_ & !LDEVSEL_
                    & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B3.OE    (ConfigAddress == CommandRegister)
                    & !LDEVSEL_
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B3.AR    !BRST_;

COMMAND_B2      := BAD2;
COMMAND_B2.CLK    (ConfigAddress == CommandRegister)
                    & !DATA_PHASE_ & !LDEVSEL_
                    & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B2.OE   = (ConfigAddress == CommandRegister)
                    & !LDEVSEL_
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B2.AR    !BRST_;

COMMAND_B1      := 0;
COMMAND_B1.CLK    (ConfigAddress == CommandRegister)
                    & !DATA_PHASE_ & !LDEVSEL_
                    & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B1.OE    (ConfigAddress == CommandRegister)
                    & !LDEVSEL_
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B1.AR    !BRST_;

COMMAND_B0      := BAD0;
COMMAND_B0.CLK    (ConfigAddress == CommandRegister)
                    & !DATA_PHASE_ & !LDEVSEL_
                    & (PCI_Command == ConfigurationWrite) & !BC_BE0_;
COMMAND_B0.OE    (ConfigAddress == CommandRegister)
                    & !LDEVSEL_
                    & (PCI_Command == ConfigurationRead) & !BC_BE0_;
COMMAND_B0.AR    !BRST_;

"Equations For Generating Valid Command Register Latched Outputs:

          IOACCS_ENABLE := COMMAND_B0.FB & BRST_;
          IOACCS_ENABLE.CLK = BCLK;
          IOACCS_ENABLE.OE    1;
          IOACCS_ENABLE.AR    !BRST_;

          MASTER_ENABLE := COMMAND_B2.FB & BRST_;
          MASTER_ENABLE.CLK   BCLK;
          MASTER_ENABLE.OE    1;
          MASTER_ENABLE.AR    !BRST_;

          MEMWIV_ENABLE := 0;
          MEMWIV_ENABLE.CLK   BCLK;
          MEMWIV_ENABLE.OE    1;
          MEMWIV_ENABLE.AR    !BRST_;

          PARERR_ENABLE := COMMAND_B6.FB & BRST_;
          PARERR_ENABLE.CLK   BCLK;
          PARERR_ENABLE.OE   1;
          PARERR_ENABLE.AR   !BRST_;

"          SYSERR_ENABLE := COMMAND_B8.FB & BRST_;
          SYSERR_ENABLE := 0;
          SYSERR_ENABLE.CLK   BCLK;
          SYSERR_ENABLE.OE    1;
          SYSERR_ENABLE.AR    !BRST_;

"Equations For Setting, Clearing And Enabling Status Register Bits:

        STATUS_B8       := MPARER_RPT & (PCI_Command == ConfigurationRead)
                           # !BAD24     & (PCI_Command == ConfigurationWrite);
        STATUS_B8.CLK       STATUS_STROBE
                           & (PCI_Command == ConfigurationRead) & !BC_BE3_
                           & STATUS_STROBE
                           & (PCI_Command == ConfigurationWrite) & !BC_BE3_;
```

```
STATUS_B8.OE       STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_;
STATUS_B8.AR     = !BRST_;

STATUS_B9          0;
STATUS_B9.OE       STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_;

STATUS_B10       = 1;
STATUS_B10.OE    =  STATUS_STROBE
                    & (PCI_Command == ConfigurationRead) & !BC_BE3_;

STATUS_B11      := STRG_ABORT & (PCI_Command == ConfigurationRead)
                 # !BAD27     & (PCI_Command == ConfigurationWrite);
STATUS_B11.CLK     STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_
                 & STATUS_STROBE
                 & (PCI_Command == ConfigurationWrite) & !BC_BE3_;
STATUS_B11.OE      STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_;
STATUS_B11.AR    = !BRST_;

STATUS_B12      := MTRG_ABORT & (PCI_Command == ConfigurationRead)
                 # !BAD28     & (PCI_Command == ConfigurationWrite);
STATUS_B12.CLK     STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_
                 & STATUS_STROBE
                 & (PCI_Command == ConfigurationWrite) & .BC_BE3_.
STATUS_B12.OE    =  STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & .BC_BE3_,
STATUS_B12.AR    = !BRST_;

STATUS_B13      := MASTER_ABORT & (PCI_Command == ConfigurationRead)
                 # !BAD29       & (PCI_Command == ConfigurationWrite);
STATUS_B13.CLK  =  STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & .BC_BE3_
                 & STATUS_STROBE
                 & (PCI_Command == ConfigurationWrite) & .BC_BE3_.
STATUS_B13.OE      STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & .BC_BE3_;
STATUS_B13.AR      .BRST_.

STATUS_B14      :=  !SERR & (PCI_Command == ConfigurationRead)
                 # !BAD30 & (PCI_Command == ConfigurationWrite);
STATUS_B14.CLK  =  STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_
                 & STATUS_STROBE
                 & (PCI_Command == ConfigurationWrite) & !BC_BE3_;
STATUS_B14.OE      STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & .BC_BE3_.
STATUS_B14.AR      !BRST_.

STATUS_B15      :=  APARER & (PCI_Command == ConfigurationRead)
                 # !BAD31  & (PCI_Command == ConfigurationWrite);
STATUS_B15.CLK     STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & !BC_BE3_
                 & STATUS_STROBE
                 & (PCI_Command == ConfigurationWrite) & !BC_BE3_;
STATUS_B15.OE      STATUS_STROBE
                 & (PCI_Command == ConfigurationRead) & .BC_BE3_.
STATUS_B15.AR      !BRST_;

STATUS_STROBE      (PCI_Command == ConfigurationRead) & !IDSEL_
                 &  (ConfigAddress == StatusRegister);
```

```
"Equations for Demultiplexing the PCI Command Signals:

CMD3     :=  BC_BE3_ & .BFRAME_ & LFRAME_
          #  CMD3 & .LFRAME_,
CMD3.CLK     .CLK66;
CMD3.OE      1;

CMD2     :=  BC_BE2_ & !BFRAME_ & LFRAME_
          #  CMD2 & !LFRAME_;
CMD2.CLK     .CLK66;
CMD2.OE      1;

CMD1     :=  BC_BE1_ & !BFRAME_ & LFRAME_
          #  CMD1 & !LFRAME_,
CMD1.CLK     .CLK66;
CMD1.OE      1;

CMD0     :=  BC_BE0_ & !BFRAME_ & LFRAME_
          #  CMD0 & !LFRAME_.
```

```
        CMD0.CLK    !CLK66;
        CMD0.OE     1;

"-------------------------------------------------------------------------------
"       Equations for Demultiplexing some Local Address Bits:
"-------------------------------------------------------------------------------
        LA7 := BAD7;
        LA7.CLK   ADDR_CLOCK;
        LA7.OE   1;

        LA6 := BAD6;
        LA6.CLK   ADDR_CLOCK;
        LA6.OE    1;

        LA5 := BAD5;
        LA5.CLK   ADDR_CLOCK;
        LA5.OE  = 1;

        LA4 := BAD4;
        LA4.CLK = ADDR_CLOCK;
        LA4.OE    1;

        LA3 := BAD3;
        LA3.CLK = ADDR_CLOCK;
        LA3.OE    1;

        LA2 := BAD2;
        LA2.CLK   ADDR_CLOCK;
        LA2.OE    1;

"-------------------------------------------------------------------------------
"                 Latency Timer Truth Table
"-------------------------------------------------------------------------------
"Output Enable Equations added for Simulation Purposes
       LTCD7.OE = SIMULATE;
       LTCD6.OE   SIMULATE;
       LTCD5.OE   SIMULATE;
       LTCD4.OE   SIMULATE;
       LTCD3.OE   SIMULATE;
       LTCD2.OE   SIMULATE;
       LTCD1.OE   SIMULATE;
       LTCD0.OE   SIMULATE;

state_diagram LTimerLo

               State 15:
                      if !MASTER_  then 14;
                      else if MASTER_ & (LTREG_OUT_LO==0) then 0
                      else if MASTER_ & (LTREG_OUT_LO==1) then 1
                      else if MASTER_ & (LTREG_OUT_LO==2) then 2
                      else if MASTER_ & (LTREG_OUT_LO==3) then 3
                      else if MASTER_ & (LTREG_OUT_LO==4) then 4
                      else if MASTER_ & (LTREG_OUT_LO==5) then 5
                      else if MASTER_ & (LTREG_OUT_LO==6) then 6
                      else if MASTER_ & (LTREG_OUT_LO==7) then 7
                      else if MASTER_ & (LTREG_OUT_LO==8) then 8
                      else if MASTER_ & (LTREG_OUT_LO==9) then 9
                      else if MASTER_ & (LTREG_OUT_LO==10) then 10
                      else if MASTER_ & (LTREG_OUT_LO==11) then 11
                      else if MASTER_ & (LTREG_OUT_LO==12) then 12
                      else if MASTER_ & (LTREG_OUT_LO==13) then 13
                      else if MASTER_ & (LTREG_OUT_LO==14) then 14
                      else               15;
               State 14:
                      if !MASTER_  then 13;
                      else if MASTER_ & (LTREG_OUT_LO==0) then 0
                      else if MASTER_ & (LTREG_OUT_LO==1) then 1
                      else if MASTER_ & (LTREG_OUT_LO==2) then 2
                      else if MASTER_ & (LTREG_OUT_LO==3) then 3
                      else if MASTER_ & (LTREG_OUT_LO==4) then 4
                      else if MASTER_ & (LTREG_OUT_LO==5) then 5
                      else if MASTER_ & (LTREG_OUT_LO==6) then 6
                      else if MASTER_ & (LTREG_OUT_LO==7) then 7
                      else if MASTER_ & (LTREG_OUT_LO==8) then 8
                      else if MASTER_ & (LTREG_OUT_LO==9) then 9
                      else if MASTER_ & (LTREG_OUT_LO==10) then 10
                      else if MASTER_ & (LTREG_OUT_LO==11) then 11
                      else if MASTER_ & (LTREG_OUT_LO==12) then 12
                      else if MASTER_ & (LTREG_OUT_LO==13) then 13
                      else if MASTER_ & (LTREG_OUT_LO==15) then 15
                      else               14;
               State 13:
                      if !MASTER_  then 12;
                      else if MASTER_ & (LTREG_OUT_LO==0) then 0
```

```
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              13;
State 12:
        if !MASTER_  then 11;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              12;
State 11:
        if !MASTER_  then 10;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              11;
State 10:
        if !MASTER_  then  9;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              10;
State 9:
        if !MASTER_   then 8;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
```

```
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              9;
State 8:
        if !MASTER_   then 7;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              8;
State 7:
        if !MASTER_   then 6;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              7;
State 6:
        if !MASTER_   then 5;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              6;
State 5:
        if !MASTER_   then 4;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else              5;
State 4:
        if !MASTER_   then 3;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
```

```
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else                4;
State 3:
        if !MASTER_    then 2;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else                3;
State 2:
        if !MASTER_    then 1;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else                2;
State 1:
        if !MASTER_    then 0;
        else if MASTER_ & (LTREG_OUT_LO==0) then 0
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else                1;
State 0:
        if !MASTER_  & (LTimerHi != 00) then 15
        else if MASTER_ & (LTREG_OUT_LO==1) then 1
        else if MASTER_ & (LTREG_OUT_LO==2) then 2
        else if MASTER_ & (LTREG_OUT_LO==3) then 3
        else if MASTER_ & (LTREG_OUT_LO==4) then 4
        else if MASTER_ & (LTREG_OUT_LO==5) then 5
        else if MASTER_ & (LTREG_OUT_LO==6) then 6
        else if MASTER_ & (LTREG_OUT_LO==7) then 7
        else if MASTER_ & (LTREG_OUT_LO==8) then 8
        else if MASTER_ & (LTREG_OUT_LO==9) then 9
        else if MASTER_ & (LTREG_OUT_LO==10) then 10
        else if MASTER_ & (LTREG_OUT_LO==11) then 11
        else if MASTER_ & (LTREG_OUT_LO==12) then 12
        else if MASTER_ & (LTREG_OUT_LO==13) then 13
        else if MASTER_ & (LTREG_OUT_LO==14) then 14
        else if MASTER_ & (LTREG_OUT_LO==15) then 15
        else                0;
```

```
state_diagram LTimerHi
                State 15:
                        If (LTimerLo==0) & !MASTER_ then 14
                        else if MASTER_ & (LTREG_OUT_HI==0) then 0
                        else if MASTER_ & (LTREG_OUT_HI==1) then 1
                        else if MASTER_ & (LTREG_OUT_HI==2) then 2
                        else if MASTER_ & (LTREG_OUT_HI==3) then 3
                        else if MASTER_ & (LTREG_OUT_HI==4) then 4
                        else if MASTER_ & (LTREG_OUT_HI==5) then 5
                        else if MASTER_ & (LTREG_OUT_HI==6) then 6
                        else if MASTER_ & (LTREG_OUT_HI==7) then 7
                        else if MASTER_ & (LTREG_OUT_HI==8) then 8
                        else if MASTER_ & (LTREG_OUT_HI==9) then 9
                        else if MASTER_ & (LTREG_OUT_HI==10) then 10
                        else if MASTER_ & (LTREG_OUT_HI==11) then 11
                        else if MASTER_ & (LTREG_OUT_HI==12) then 12
                        else if MASTER_ & (LTREG_OUT_HI==13) then 13
                        else if MASTER_ & (LTREG_OUT_HI==14) then 14
                        else 15;
                State 14:
                        If (LTimerLo==0) & !MASTER_ then 13
                        else if MASTER_ & (LTREG_OUT_HI==0) then 0
                        else if MASTER_ & (LTREG_OUT_HI==1) then 1
                        else if MASTER_ & (LTREG_OUT_HI==2) then 2
                        else if MASTER_ & (LTREG_OUT_HI==3) then 3
                        else if MASTER_ & (LTREG_OUT_HI==4) then 4
                        else if MASTER_ & (LTREG_OUT_HI==5) then 5
                        else if MASTER_ & (LTREG_OUT_HI==6) then 6
                        else if MASTER_ & (LTREG_OUT_HI==7) then 7
                        else if MASTER_ & (LTREG_OUT_HI==8) then 8
                        else if MASTER_ & (LTREG_OUT_HI==9) then 9
                        else if MASTER_ & (LTREG_OUT_HI==10) then 10
                        else if MASTER_ & (LTREG_OUT_HI==11) then 11
                        else if MASTER_ & (LTREG_OUT_HI==12) then 12
                        else if MASTER_ & (LTREG_OUT_HI==13) then 13
                        else if MASTER_ & (LTREG_OUT_HI==15) then 15
                        else 14;
                State 13:
                        If (LTimerLo==0) & !MASTER_ then 12
                        else if MASTER_ & (LTREG_OUT_HI==0) then 0
                        else if MASTER_ & (LTREG_OUT_HI==1) then 1
                        else if MASTER_ & (LTREG_OUT_HI==2) then 2
                        else if MASTER_ & (LTREG_OUT_HI==3) then 3
                        else if MASTER_ & (LTREG_OUT_HI==4) then 4
                        else if MASTER_ & (LTREG_OUT_HI==5) then 5
                        else if MASTER_ & (LTREG_OUT_HI==6) then 6
                        else if MASTER_ & (LTREG_OUT_HI==7) then 7
                        else if MASTER_ & (LTREG_OUT_HI==8) then 8
                        else if MASTER_ & (LTREG_OUT_HI==9) then 9
                        else if MASTER_ & (LTREG_OUT_HI==10) then 10
                        else if MASTER_ & (LTREG_OUT_HI==11) then 11
                        else if MASTER_ & (LTREG_OUT_HI==12) then 12
                        else if MASTER_ & (LTREG_OUT_HI==14) then 14
                        else if MASTER_ & (LTREG_OUT_HI==15) then 15
                        else 13;
                State 12:
                        If (LTimerLo==0) & !MASTER_ then 11
                        else if MASTER_ & (LTREG_OUT_HI==0) then 0
                        else if MASTER_ & (LTREG_OUT_HI==1) then 1
                        else if MASTER_ & (LTREG_OUT_HI==2) then 2
                        else if MASTER_ & (LTREG_OUT_HI==3) then 3
                        else if MASTER_ & (LTREG_OUT_HI==4) then 4
                        else if MASTER_ & (LTREG_OUT_HI==5) then 5
                        else if MASTER_ & (LTREG_OUT_HI==6) then 6
                        else if MASTER_ & (LTREG_OUT_HI==7) then 7
                        else if MASTER_ & (LTREG_OUT_HI==8) then 8
                        else if MASTER_ & (LTREG_OUT_HI==9) then 9
                        else if MASTER_ & (LTREG_OUT_HI==10) then 10
                        else if MASTER_ & (LTREG_OUT_HI==11) then 11
                        else if MASTER_ & (LTREG_OUT_HI==13) then 13
                        else if MASTER_ & (LTREG_OUT_HI==14) then 14
                        else if MASTER_ & (LTREG_OUT_HI==15) then 15
                        else 12;
                State 11:
                        If (LTimerLo==0) & !MASTER_ then 10
                        else if MASTER_ & (LTREG_OUT_HI==0) then 0
                        else if MASTER_ & (LTREG_OUT_HI==1) then 1
                        else if MASTER_ & (LTREG_OUT_HI==2) then 2
                        else if MASTER_ & (LTREG_OUT_HI==3) then 3
                        else if MASTER_ & (LTREG_OUT_HI==4) then 4
                        else if MASTER_ & (LTREG_OUT_HI==5) then 5
                        else if MASTER_ & (LTREG_OUT_HI==6) then 6
                        else if MASTER_ & (LTREG_OUT_HI==7) then 7
                        else if MASTER_ & (LTREG_OUT_HI==8) then 8
```

```
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 11;
State 10:
        If (LTimerLo==0) & !MASTER_ then 9
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 10;
State 9:
        If (LTimerLo==0) & !MASTER_ then 8
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 9;
State 8:
        If (LTimerLo==0) & !MASTER_ then 7
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 8;
State 7:
        If (LTimerLo==0) & !MASTER_ then 6
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 7;
State 6:
        If (LTimerLo==0) & !MASTER_ then 5
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
```

```
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 6;
State 5:
        If (LTimerLo==0) & !MASTER_ then 4
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 5;
State 4:
        If (LTimerLo==0) & !MASTER_ then 3
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 4;
State 3:
        If (LTimerLo==0) & !MASTER_ then 2
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 3;
State 2:
        If (LTimerLo==0) & !MASTER_ then 1
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
```

```
        else 2;
State 1:
        If (LTimerLo==0) & !MASTER_ then 0
        else if MASTER_ & (LTREG_OUT_HI==0) then 0
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 1;
State 0:
        If !MASTER_ then 0
        else if MASTER_ & (LTREG_OUT_HI==1) then 1
        else if MASTER_ & (LTREG_OUT_HI==2) then 2
        else if MASTER_ & (LTREG_OUT_HI==3) then 3
        else if MASTER_ & (LTREG_OUT_HI==4) then 4
        else if MASTER_ & (LTREG_OUT_HI==5) then 5
        else if MASTER_ & (LTREG_OUT_HI==6) then 6
        else if MASTER_ & (LTREG_OUT_HI==7) then 7
        else if MASTER_ & (LTREG_OUT_HI==8) then 8
        else if MASTER_ & (LTREG_OUT_HI==9) then 9
        else if MASTER_ & (LTREG_OUT_HI==10) then 10
        else if MASTER_ & (LTREG_OUT_HI==11) then 11
        else if MASTER_ & (LTREG_OUT_HI==12) then 12
        else if MASTER_ & (LTREG_OUT_HI==13) then 13
        else if MASTER_ & (LTREG_OUT_HI==14) then 14
        else if MASTER_ & (LTREG_OUT_HI==15) then 15
        else 0;


end DECODE
```

5.3.4. PARIO PLD

```
module PARIO flag'-r3'
title 'Parallel I/O Interface Controller for PCI Based PCI I/O board
Sheet 6 of Schematic Diagrams
"-------------------------------------------------------------------------------------------
"  This PLD implements the following functionality on the PCI Parallel
"  Interface Add-In Board:
"
"    Implements the Parallel Interface State Machine which controls data access to the
"    4K byte Input/Output FIFO Buffers.  The Input/Output FIFO Buffers are 32-Bits
"    wide on the PCI Interface Side and 8-Bits wide on the I/O Side.
"    This PLD provides fourteen (14) Control pins which may be defined by the user.  This
"    enables the designer to implement a complete custom interface
"    without adding any additional devices other than the required drivers for the
"    interface selected.  In some cases the signal CONTROLx is inserted into equations to
"    indicate that one or more of the interface control strobes should be used to control
"    the particular equation.  This design file may be used as a template, but it is up to the
"    designer to make sure that the interface can be properly implemented in this device.
"-------------------------------------------------------------------------------------------
"                  Declarations
"-------------------------------------------------------------------------------------------

                   PARIO      device 'ifx780fp' ;

"-------------------------------------------------------------------------------------------
"                  Local PCI Interface Input Pins
"-------------------------------------------------------------------------------------------
       BCLK                    pin 118;  "Buffered (up to 33Mhz) PCI Bus Clock
       BRST_                   pin   1;  "Local Active Low PCI Reset
       LA2                     pin   2;  "Local Demultiplexed Address Bit 2
       LA3                     pin   3;  "Local Demultiplexed Address Bit 3
       LA4                     pin   4;  "Local Demultiplexed Address Bit 4
       LA5                     pin   5;  "Local Demultiplexed Address Bit 5
       LA6                     pin  33;  "Local Demultiplexed Address Bit 6
       LA7                     pin  34;  "Local Demultiplexed Address Bit 7
       I_CTL_WR_               pin  35;  "IO Control Address Space Write#
       I_STA_RD_               pin  36;  "IO Status Address Space Read#
       DMA_ACK                 pin  37;  "DMA Cycle Acknowledge
"-------------------------------------------------------------------------------------------
"                  Local PCI Interface Output Pins
"-------------------------------------------------------------------------------------------
       PAR_INTRQ               pin 106;  "IO Interface PCI Interrupt Request
       DMA_REQUEST_            pin  78;  "FIFO DMA Request#
       DMA_BREQUEST_           pin  47;  "FIFO DMA Burst Request#

"-------------------------------------------------------------------------------------------
"                  Parallel Interface Input Pins
"-------------------------------------------------------------------------------------------

       I_WR_STROBE_            pin  96;  "Parallel Interface Byte Write Strobe#
       I_RD_STROBE_            pin  38;  "Parallel Interface Byte Read  Strobe#
       I_RD_STBAGN_            pin  67;  "Parallel Interface Byte Read Clock (I_RD_STROBE_)
       I_WR_STROBES_           pin  63;  "Parallel Byte Write Strobe Synced to BCLK#
       I_RD_STROBES_           pin  64;  "Parallel Byte Read Strobe Synced to BCLK#

"-------------------------------------------------------------------------------------------
"                  Parallel Interface Output Pins
"-------------------------------------------------------------------------------------------

I_EMPTY_               pin  79;  "Parallel Interface Buffer Empty Status#
       I_FULL_                 pin  95;  "Parallel Interface Buffer Half Status#
       I_HALF_FULL_            pin  80;  "Parallel Interface Buffer Half Full Status#
I_DEBUG                node 32;  "Parallel Interface Buried Debug Control
       I_REQUEST               node 31;  "Parallel Interface Buried Request Control
                                         "This is a Software control pin
                                         "used for diagnostic purposes.
"-------------------------------------------------------------------------------------------
"                  Parallel Interface User Defined Pins
"-------------------------------------------------------------------------------------------
       I_CONTROL1              pin   7;  "Parallel Interface User Defined Control 1
       I_CONTROL2              pin  14;  "Parallel Interface User Defined Control 2
       I_CONTROL3              pin  68;  "Parallel Interface User Defined Control 3
       I_CONTROL4              pin  69;  "Parallel Interface User Defined Control 4
       I_CONTROL5              pin 109;  "Parallel Interface User Defined Control 5
       I_CONTROL6              pin  24;  "Parallel Interface User Defined Control 6
       I_CONTROL7              pin  92;  "Parallel Interface User Defined Control 7
       I_CONTROL8              pin  94;  "Parallel Interface User Defined Control 8
       I_CONTROL9              pin  25;  "Input Status Connection from Scanner or Marker
       I_CONTROL10             pin  91;  "Parallel Interface User Defined Control 9
       I_CONTROL11             pin 108;  "Parallel Interface User Defined Control 11
       I_CONTROL12             pin 107;  "Parallel Interface User Defined Control 12
       I_CONTROL13             pin   8;  "Parallel Interface User Defined Control 13
       I_CONTROL14             pin 114;  "Parallel Interface User Defined Control 14
```

```
"-------------------------------------------------------------------------------------------------
"                  FIFO Interface Input Pins
"-------------------------------------------------------------------------------------------------

        IFIFOB0_FULL_           pin  98;  "Input FIFO Byte 0 Full Status#
        IFIFOB1_FULL_           pin  97;  "Input FIFO Byte 1 Full Status#
        IFIFOB2_FULL_           pin  70;  "Input FIFO Byte 2 Full Status#
        IFIFOB3_FULL_           pin  71;  "Input FIFO Byte 3 Full Status#
        IFIFOB0_EMPTY_          pin  99;  "Input FIFO Byte 0 Empty Status#
        IFIFOB1_EMPTY_          pin 100;  "Input FIFO Byte 1 Empty Status#
        IFIFOB2_EMPTY_          pin 101;  "Input FIFO Byte 2 Empty Status#
        IFIFOB3_EMPTY_          pin 102;  "Input FIFO Byte 3 Empty Status#
        IFIFOB0_HALFF_          pin 103;  "Input FIFO Byte 0 Half Full Status#
        IFIFOB1_HALFF_          pin 104;  "Input FIFO Byte 1 Half Full Status#
        IFIFOB2_HALFF_          pin  32;  "Input FIFO Byte 2 Half Full Status#
        IFIFOB3_HALFF_          pin  31;  "Input FIFO Byte 3 Half Full Status#
        IFIFOB3_AF_AEB          pin   6;  "Input FIFO Byte 3 Almost Full Status#


        OFIFOB0_FULL_           pin  13;  "Output FIFO Byte 0 Full Status#
        OFIFOB1_FULL_           pin  16;  "Output FIFO Byte 1 Full Status#
        OFIFOB2_FULL_           pin 105;  "Output FIFO Byte 2 Full Status#
        OFIFOB3_FULL_           pin 115;  "Output FIFO Byte 3 Full Status#
        OFIFOB0_EMPTY_          pin  39;  "Output FIFO Byte 0 Empty Status#
        OFIFOB1_EMPTY_          pin  40;  "Output FIFO Byte 1 Empty Status#
        OFIFOB2_EMPTY_          pin  41;  "Output FIFO Byte 2 Empty Status#
        OFIFOB3_EMPTY_          pin  42;  "Output FIFO Byte 3 Empty Status#
        OFIFOB0_HALFF_          pin  43;  "Output FIFO Byte 0 Half Full Status#
        OFIFOB1_HALFF_          pin  45;  "Output FIFO Byte 1 Half Full Status#
        OFIFOB2_HALFF_          pin  72;  "Output FIFO Byte 2 Half Full Status#
        OFIFOB3_HALFF_          pin  73;  "Output FIFO Byte 3 Half Full Status#
        OFIFOB3_AF_AEA          pin  74;  "Output FIFO Byte 3 Almost Full Status#

"-------------------------------------------------------------------------------------------------
"                  FIFO Interface Output Pins
"-------------------------------------------------------------------------------------------------

        OFIFOB0_RD_             pin 124;  "Output FIFO Byte 0 Read Strobe#
        OFIFOB1_RD_             pin 127;  "Output FIFO Byte 1 Read Strobe#
        OFIFOB2_RD_             pin 128;  "Output FIFO Byte 2 Read Strobe#
        OFIFOB3_RD_             pin 126;  "Output FIFO Byte 3 Read Strobe#

        IFIFOB0_WR_             pin 129;  "Input FIFO Byte 0 Write Strobe#
        IFIFOB1_WR_             pin 122;  "Input FIFO Byte 1 Write Strobe#
        IFIFOB2_WR_             pin 123;  "Input FIFO Byte 2 Write Strobe#
        IFIFOB3_WR_             pin 121;  "Input FIFO Byte 3 Write Strobe#

"-------------------------------------------------------------------------------------------------
"        PCI Bidirectional Local Data Bits LSB
"-------------------------------------------------------------------------------------------------

        BAD7                     pin  89; "Local Data Bit 7
        BAD6                     pin  82; "Local Data Bit 6
        BAD5                     pin  88; "Local Data Bit 5
        BAD4                     pin  81; "Local Data Bit 4
        BAD3                     pin  90; "Local Data Bit 3
        BAD2                     pin  49; "Local Data Bit 2
        BAD1                     pin  22; "Local Data Bit 1
        BAD0                     pin  48; "Local Data Bit 0

"-------------------------------------------------------------------------------------------------
"   Parallel Interface Bidirectional Parallel Data Bus (Data Connection To Transceivers)
"-------------------------------------------------------------------------------------------------

        IPD7                     pin  10; "Parallel Data Bit 7
        IPD7_IN                  pin  10; "Parallel Data Bit 7
        IPD6                     pin  28; "Parallel Data Bit 6
        IPD6_IN                  pin  28; "Parallel Data Bit 6
        IPD5                     pin  30; "Parallel Data Bit 5
        IPD5_IN                  pin  30; "Parallel Data Bit 5
        IPD4                     pin  29; "Parallel Data Bit 4
        IPD4_IN                  pin  29; "Parallel Data Bit 4
        IPD3                     pin   9; "Parallel Data Bit 3
        IPD3_IN                  pin   9; "Parallel Data Bit 3
        IPD2                     pin  76; "Parallel Data Bit 2
```

```
        IPD2_IN                 pin  76; "Parallel Data Bit 2
        IPD1                    pin  26; "Parallel Data Bit 1
        IPD1_IN                 pin  26; "Parallel Data Bit 1
        IPD0                    pin  46; "Parallel Data Bit 0
        IPD0_IN                 pin  46; "Parallel Data Bit 0

"-------------------------------------------------------------------------------------------------
"   Parallel Interface Bidirectional Parallel Data Bus (Data Connection To FIFOs)
"-------------------------------------------------------------------------------------------------

        FPD7                    pin  54; "Parallel Data Bit 7
        FPD6                    pin  58; "Parallel Data Bit 6
        FPD5                    pin  55; "Parallel Data Bit 5
        FPD4                    pin  60; "Parallel Data Bit 4
        FPD3                    pin  56; "Parallel Data Bit 3
        FPD2                    pin  61; "Parallel Data Bit 2
        FPD1                    pin  57; "Parallel Data Bit 1
        FPD0                    pin  62; "Parallel Data Bit 0

"-------------------------------------------------------------------------------------------------
"        Parallel Interface State Machine Internal (Buried) State Registers
"-------------------------------------------------------------------------------------------------

        FIFO_OP                 node  16; "Parallel Interface FIFO Operation Flag
        PAR_S3                  node  13; "Parallel Interface Internal State Register 3
        PAR_S2                  node   6; "Parallel Interface Internal State Register 2
        PAR_S1                  node 105; "Parallel Interface Internal State Register 1
        PAR_S0                  node 115; "Parallel Interface Internal State Register 0

"-------------------------------------------------------------------------------------------------
"                  FIFO Buffer Mode Control Outputs
"-------------------------------------------------------------------------------------------------

        FIFO_SAB                pin 130; "FIFO A to B Transceiver Mode Control
        FIFO_SBA                pin 112; "FIFO B to A Transceiver Mode Control
        FIFO_B3GAB              pin 120; "FIFO Byte 3 Data Output Enable Control
        FIFO_B2GAB              pin 113; "FIFO Byte 2 Data Output Enable Control
        FIFO_B1GAB              pin  15; "FIFO Byte 1 Data Output Enable Control
        FIFO_B0GAB              pin 111; "FIFO Byte 0 Data Output Enable Control

"-------------------------------------------------------------------------------------------------
"                  Outputs and Feedback Terms
"-------------------------------------------------------------------------------------------------

        IFIFO_DBF               pin  75; "Input FIFO Define B Flag Strobe#
        PDBUS_ENABLE            pin  23; "P-Term to Enable driving PD Bus

        P_DATACLKO              pin  12 ; "Combinatorial Clock Eq to Drive Data
                                          "onto IPD Bus
        P_DATACLKI              pin 52; "Clock Input to Drive Data onto IPD Bus

"-------------------------------------------------------------------------------------------------
"                  Set Definitions and Constants
"-------------------------------------------------------------------------------------------------

        H,L,X,C,K,Z,SIMULATE   1,0,.X.,.C.,.K.,.Z.,0;

        I_DEBUG         istype 'buffer, pos, reg';
        I_REQUEST       istype 'buffer, pos, reg';
        FIFO_OP         istype 'buffer, pos, reg';
        PAR_S3          istype 'buffer, pos, reg';
        PAR_S2          istype 'buffer, pos, reg';
        PAR_S1          istype 'buffer, pos, reg';
        PAR_S0          istype 'buffer, pos, reg';

        IO_AddrIn       [ LA5 , LA4 , LA3 , LA2 ];

        IO_Address      [ LA7 , LA6 , LA5 , LA4 , LA3 , LA2 ];
        Status          [  X  ,  0  ,  0  ,  0  ,  0  ,  0  ];
        Vector          [  X  ,  0  ,  0  ,  0  ,  0  ,  1  ];
        LoopBackOut     [  X  ,  0  ,  0  ,  1  ,  X  ,  X  ];
        LoopBackOut0    [  X  ,  0  ,  0  ,  1  ,  0  ,  0  ];
        LoopBackOut1    [  X  ,  0  ,  0  ,  1  ,  0  ,  1  ];
        LoopBackOut2    [  X  ,  0  ,  0  ,  1  ,  1  ,  0  ];
        LoopBackOut3    [  X  ,  0  ,  0  ,  1  ,  1  ,  1  ];
        LoopBackIn      [  X  ,  0  ,  1  ,  0  ,  X  ,  X  ];
        LoopBackIn0     [  X  ,  0  ,  1  ,  0  ,  0  ,  0  ];
        LoopBackIn1     [  X  ,  0  ,  1  ,  0  ,  0  ,  1  ];
        LoopBackIn2     [  X  ,  0  ,  1  ,  0  ,  1  ,  0  ];
        LoopBackIn3     [  X  ,  0  ,  1  ,  0  ,  1  ,  1  ];
        FIFO_Status0 =  [  X  ,  0  ,  1  ,  1  ,  0  ,  0  ];
        FIFO_Status1    [  X  ,  0  ,  1  ,  1  ,  0  ,  1  ];
        FIFO_Status2    [  X  ,  0  ,  1  ,  1  ,  1  ,  0  ];
        FIFO_Status3    [  X  ,  0  ,  1  ,  1  ,  1  ,  1  ];
```

```
        IFIFO_FLAG      [  X  ,  1  ,  0  ,  1  ,  0  ,  1  ];

        IO_Mode         [ ,I_HS_MODE];

        FIFO_Input      [    0    ,    0    ];
        FIFO_Output  =  [    1    ,    0    ];
        HS_Input        [    0    ,    1    ];
        HS_Output    -  [    1    ,    1    ];

        IFIFO_Status    [IFIFOB3_FULL_ ,IFIFOB2_FULL_ ,IFIFOB1_FULL_ ,IFIFOB0_FULL_,
                         IFIFOB3_EMPTY_,IFIFOB2_EMPTY_,IFIFOB1_EMPTY_,IFIFOB0_EMPTY_,
                         IFIFOB3_HALFF_,IFIFOB2_HALFF_,IFIFOB1_HALFF_,IFIFOB0_HALFF_,
                         IFIFOB3_AF_AEB,IFIFOB3_AF_AEB,IFIFOB3_AF_AEB,IFIFOB3_AF_AEB];

        OFIFO_Status    [OFIFOB3_FULL_ ,OFIFOB2_FULL_ ,OFIFOB1_FULL_ ,OFIFOB0_FULL_,
                         OFIFOB3_EMPTY_,OFIFOB2_EMPTY_,OFIFOB1_EMPTY_,OFIFOB0_EMPTY_,
                         OFIFOB3_HALFF_,OFIFOB2_HALFF_,OFIFOB1_HALFF_,OFIFOB0_HALFF_,
                         OFIFOB3_AF_AEA,OFIFOB3_AF_AEA,OFIFOB3_AF_AEA,OFIFOB3_AF_AEA];

        Full            [ 0, 0, 0, 0, X, X, X, X, X, X, X, X, X, X, X, X ];
        Empty           [ X, X, X, X, 0, 0, 0, 0, X, X, X, X, X, X, X, X ];
        NotEmpty        [ X, X, X, X, 1, 1, 1, 1, X, X, X, X, X, X, X, X ];
        HalfFull        [ X, X, X, X, X, X, X, X, 1, 1, 1, 1, X, X, X, X ];
        AlmostEmpty     [ X, X, X, X, X, X, X, X, 0, X, X, X, 1, X, X, X ];
        AlmostFull      [ X, X, X, X, X, X, X, X, 1, X, X, X, 1, X, X, X ];
"-------------------------------------------------------------------------------
"FIFO Status operates as follows:
"     When the FIFO is Almost Empty or Almost Full bits(7&3) are HIGH.
"     When the FIFO is Half or More Full bits (6&2) are HIGH.
"     When the FIFO is FULL bits (5&1) are INVERTED to be HIGH.
"     When the FIFO is EMPTY bits (4&0) are INVERTED to be HIGH.
"-------------------------------------------------------------------------------
        FIFOB0_Status = [IFIFOB3_AF_AEB,IFIFOB0_HALFF_,!IFIFOB0_FULL_,!IFIFOB0_EMPTY_,
                         OFIFOB3_AF_AEA,OFIFOB0_HALFF_,!OFIFOB0_FULL_,!OFIFOB0_EMPTY_];

        FIFOB1_Status = [IFIFOB3_AF_AEB,IFIFOB1_HALFF_,!IFIFOB1_FULL_,!IFIFOB1_EMPTY_,
                         OFIFOB3_AF_AEA,OFIFOB1_HALFF_,!OFIFOB1_FULL_,!OFIFOB1_EMPTY_];

        FIFOB2_Status = [IFIFOB3_AF_AEB,IFIFOB2_HALFF_,!IFIFOB2_FULL_,!IFIFOB2_EMPTY_,
                         OFIFOB3_AF_AEA,OFIFOB2_HALFF_,!OFIFOB2_FULL_,!OFIFOB2_EMPTY_];

        FIFOB3_Status   [IFIFOB3_AF_AEB,IFIFOB3_HALFF_,!IFIFOB3_FULL_,!IFIFOB3_EMPTY_,
                         OFIFOB3_AF_AEA,OFIFOB3_HALFF_,!OFIFOB3_FULL_,!OFIFOB3_EMPTY_];

        IVector         [BAD7..BAD0];

        Local_Data   =  [BAD7..BAD0];

        FIFO_Data       [FD7..FD0];

"-------------------------------------------------------------------------------------------------
"                   Parallel Interface State Machine
"-------------------------------------------------------------------------------------------------
"  User defined state machine would be inserted here.


"-------------------------------------------------------------------------------------------------
"                   Equations
"-------------------------------------------------------------------------------------------------
equations

"equations for generating Interface FIFO status signals

       !I_EMPTY_ := (IFIFO_Status == Empty);
        I_EMPTY_.CLK  = BCLK;
        I_EMPTY_.AR   = !BRST_;
        I_EMPTY_.OE     1;

       !I_FULL_  := (IFIFO_Status == Full);
        I_FULL_.CLK   BCLK;
        I_FULL_.AR    !BRST_;
        I_FULL_.OE    1;

       !I_HALF_FULL_  := (IFIFO_Status == HalfFull);
        I_HALF_FULL_.CLK = BCLK;
        I_HALF_FULL_.AR    !BRST_;
        I_HALF_FULL_.OE    1;

        I_REQUEST := BAD7 & BRST_;
        I_REQUEST.CLK   !I_CTL_WR_ & (IO_Address == Status);
        I_REQUEST.AR    !BRST_;
        I_REQUEST.OE    SIMULATE;

        I_DEBUG   := BAD5 & BRST_,
```

```
        I_DEBUG.CLK    !I_CTL_WR_ & (IO_Address == Status);
        I_DEBUG.AR     !BRST_;
        I_DEBUG.OE     SIMULATE;


"equations for accessing internal registers

        BAD7       I_REQUEST       & (IO_Address == Status)
                # IFIFOB3_AF_AEB    & (IO_Address == FIFO_Status0)
                # IFIFOB3_AF_AEB    & (IO_Address == FIFO_Status1)
                # IFIFOB3_AF_AEB    & (IO_Address == FIFO_Status2)
                # IFIFOB3_AF_AEB    & (IO_Address == FIFO_Status3)
                # PD7               & (IO_Address == LoopBackOut);
        BAD7.OE    !IO_STA_RD_;

        BAD6     = I_CONTROLx      & (IO_Address == Status)
                # IFIFOB0_HALFF_    & (IO_Address == FIFO_Status0)
                # IFIFOB1_HALFF_    & (IO_Address == FIFO_Status1)
                # IFIFOB2_HALFF_    & (IO_Address == FIFO_Status2)
                # IFIFOB3_HALFF_    & (IO_Address == FIFO_Status3)
                # PD6               & (IO_Address == LoopBackOut);
        BAD6.OE = !IO_STA_RD_;

        BAD5       I_CONTROLx      & (IO_Address == Status)
                # !IFIFOB0_FULL_    & (IO_Address == FIFO_Status0)
                # !IFIFOB1_FULL_    & (IO_Address == FIFO_Status1)
                # !IFIFOB2_FULL_    & (IO_Address == FIFO_Status2)
                # !IFIFOB3_FULL_    & (IO_Address == FIFO_Status3)
                # PD5               & (IO_Address == LoopBackOut);
        BAD5.OE     !IO_STA_RD_;

        BAD4     = I_CONTROLx       & (IO_Address == Status)
                # !IFIFOB0_EMPTY_   & (IO_Address == FIFO_Status0)
                # !IFIFOB1_EMPTY_   & (IO_Address == FIFO_Status1)
                # !IFIFOB2_EMPTY_   & (IO_Address == FIFO_Status2)
                # !IFIFOB3_EMPTY_   & (IO_Address == FIFO_Status3)
                # PD4               & (IO_Address == LoopBackOut);
        BAD4.OE     !IO_STA_RD_;

        BAD3       I_CONTROLx      & (IO_Address == Status)
                # OFIFOB3_AF_AEA    & (IO_Address == FIFO_Status0)
                # OFIFOB3_AF_AEA    & (IO_Address == FIFO_Status1)
                # OFIFOB3_AF_AEA    & (IO_Address == FIFO_Status2)
                # OFIFOB3_AF_AEA    & (IO_Address == FIFO_Status3)
                # PD3               & (IO_Address == LoopBackOut);
        BAD3.OE     !IO_STA_RD_;

        BAD2       I_CONTROLx      & (IO_Address == Status)
                # OFIFOB0_HALFF_    & (IO_Address == FIFO_Status0)
                # OFIFOB1_HALFF_    & (IO_Address == FIFO_Status1)
                # OFIFOB2_HALFF_    & (IO_Address == FIFO_Status2)
                # OFIFOB3_HALFF_    & (IO_Address == FIFO_Status3)
                # PD2               & (IO_Address == LoopBackOut);
        BAD2.OE    !IO_STA_RD_;

        BAD1     =  I_CONTROLx      & (IO_Address == Status)
                # !OFIFOB0_FULL_    & (IO_Address == FIFO_Status0)
                # !OFIFOB1_FULL_    & (IO_Address == FIFO_Status1)
                # !OFIFOB2_FULL_    & (IO_Address == FIFO_Status2)
                # !OFIFOB3_FULL_    & (IO_Address == FIFO_Status3)
                #  PD1              & (IO_Address == LoopBackOut);
        BAD1.OE = !IO_STA_RD_;

        BAD0     =  I_CONTROLx      & (IO_Address == Status)
                # !OFIFOB0_EMPTY_   & (IO_Address == FIFO_Status0)
                # !OFIFOB1_EMPTY_   & (IO_Address == FIFO_Status1)
                # !OFIFOB2_EMPTY_   & (IO_Address == FIFO_Status2)
                # !OFIFOB3_EMPTY_   & (IO_Address == FIFO_Status3)
                #  PD0              & (IO_Address == LoopBackOut);
        BAD0.OE = !IO_STA_RD_;

"Equations to allow programmming of the Input FIFO Almost Empty/Almost Full Flag

        PDBUS_ENABLE    =  IFIFO_DBF & !CONTROLX & !IO_CTL_WR_ & (IO_Address == IFIFO_FLAG)
                        # !IO_CTL_WR_ & (IO_Address == LoopBackIn)
                        # CONTROLX & ! ,

        PD7      = BAD7 & !CONTROLX
                # IPD7_IN &  CONTROLX;
        PD7.OE     PDBUS_ENABLE;

        PD6        BAD6 & !CONTROLX
                # IPD6_IN &  CONTROLX;
        PD6.OE     PDBUS_ENABLE;
```

```
PD5       BAD5 & !CONTROLX
        # IPD5_IN &  CONTROLX;
PD5.OE    PDBUS_ENABLE;

PD4     = BAD4 & !CONTROLX
        # IPD4_IN &  CONTROLX;
PD4.OE    PDBUS_ENABLE;

PD3       BAD3 & !CONTROLX
        # IPD3_IN &  CONTROLX;
PD3.OE    PDBUS_ENABLE;

PD2       BAD2 & !CONTROLX
        # IPD2_IN &  CONTROLX;
PD2.OE    PDBUS_ENABLE;

PD1     = BAD1 & !CONTROLX
        # IPD1_IN &  CONTROLX;
PD1.OE    PDBUS_ENABLE;

PD0     = BAD0 & !CONTROLX
        # IPD0_IN &  CONTROLX;
PD0.OE    PDBUS_ENABLE;


IPD7    := PD7;
IPD7.CLK = P_DATACLKI;
IPD7.OE  =  CONTROLx.fb;

IPD6    := PD6;
IPD6.CLK = P_DATACLKI;
IPD6.OE  = CONTROLx.fb;

IPD5    := PD5;
IPD5.CLK = P_DATACLKI;
IPD5.OE  = CONTROLx.fb;

IPD4    := PD4;
IPD4.CLK = P_DATACLKI;
IPD4.OE  = CONTROLx.fb;

IPD3    := PD3;
IPD3.CLK = P_DATACLKI;
IPD3.OE    CONTROLx.fb;

IPD2    := PD2;
IPD2.CLK   P_DATACLKI;
IPD2.OE    CONTROLx.fb;

IPD1    := PD1;
IPD1.CLK   P_DATACLKI;
IPD1.OE    CONTROLx.fb;

IPD0    := PD0;
IPD0.CLK   P_DATACLKI;
IPD0.OE    CONTROLx.fb;

"Clock, Output Enable and Reset Equations for Parallel Interface State Machine


FIFO_OP.CLK    BCLK;
FIFO_OP.AR     !BRST_;
FIFO_OP.OE     SIMULATE;

PAR_S3.CLK  = BCLK;
PAR_S3.AR     !BRST_;
PAR_S3.OE     SIMULATE;

PAR_S2.CLK    BCLK;
PAR_S2.AR     !BRST_;
PAR_S2.OE     SIMULATE;

PAR_S1.CLK    BCLK;
PAR_S1.AR     !BRST_;
PAR_S1.OE   = SIMULATE;

PAR_S0.CLK  = BCLK;
PAR_S0.AR   = !BRST_;
PAR_S0.OE   = SIMULATE;

"Equations for FIFO Input/Output on Parallel Interface Side
```

```
       "OFIFOBx_RD_  signals are based upon control signals and State Machine Transitions
       "IFIFOBx_WR_  signals are based upon control signals and State Machine Transitions

"Equations for generating DMA Requests to DMA Controller on PCI Side

         DMA_BREQUEST_.CLK   BCLK;
         DMA_BREQUEST_.OE    1;
         DMA_BREQUEST_.ASET   !BRST_;

        !DMA_BREQUEST_ := (OFIFO_Status != AlmostFull)  & !DMA_BREQUEST_
                       #  (OFIFO_Status == AlmostEmpty) &  DMA_BREQUEST_
                       #  (IFIFO_Status != AlmostEmpty) & !DMA_BREQUEST_
                       #  (IFIFO_Status == AlmostFull)  &  DMA_BREQUEST_;

         DMA_REQUEST_.CLK     BCLK;
         DMA_REQUEST_.OE      1;
         DMA_REQUEST_.ASET   !BRST_;
        !DMA_REQUEST_      :=  (IFIFO_Status == NotEmpty) & !I_CONTROLx
                            #  (IFIFO_Status == NotEmpty) & !I_CONTROLx;

"Equations for generating Parallel Interface Interrupt Requests

        PAR_INTRQ    := !I_CONTROLx;
        PAR_INTRQ.CLK =  BCLK;
        PAR_INTRQ.OE     1;

"Equations for synchronizing Parallel Interface FIFO Read and Write Strobes to BCLK

        !I_WR_STROBES_ := !I_WR_STROBE_;
        !I_RD_STROBES_ := !I_RD_STROBE_;

         I_WR_STROBES_.CLK = BCLK;
         I_WR_STROBES_.OE  = 1;

         I_RD_STROBES_.CLK = BCLK;
         I_RD_STROBES_.OE  = 1;


"Equations for controlling FIFO transceivers mode and direction

         "FIFO_B0GABx signals would be controlled based upon state machine design;

         FIFO_SAB     1;

         FIFO_SBA     1;

end PARIO
```

5.4. BIBLIOGRAPHY

Austin, Connie, and Giuseppe Marescalco. "The PCI (Peripheral Component Interconnect) Local Bus." Olivetti Tech Journal, Issue 27 - January 1995.

Boynton, John. "PCI bus primed for a 64-bit extension." Electronic Engineering Times Interactive, Issue 897, page 56. April 15, 1996.

Britton, Barry, and Eirck Cook. "Design An FPGA-Based PCI Bus Interface." Electronic Design, pp.100-105. March 6, 1995.

Clarke, Michelle. "PCI bus to foreg VME connection." Electronic Engineering Times Interactive, Issue 794, page 1. April 25, 1994.

Clarke, Michelle. "PCI proliferates profusely." Electronic Engineering Times Interactive, Issue 821, page 46. October 31, 1994.

Clarke, Michele. "PCI bus gets 'switched' to 132Mbytes/s." Electronic Engineering Times, p.1, p.8. March 20, 1995.

Cole, Bernard C.. "LSI Logic connects PCI bus to FlexCore ASICs." Electronic Engineering Times Interactive, Issue 807, page 102. July 25, 1994.

Cole, Bernard C.. "Choices wide open in PC-graphics design." Electronic Engineering Times, pp. 44-49, p.59, p.73. August 7, 1995.

Costlow, Terry. "Industrial market warms up to PCI." Electronic Engineering Times, p.4. October 9, 1995.

Creede, Frank, and Erick Cook. "An FPGA-Based PCI Bus Interface." Integrated System Design, pp.32-44. April 1995.

Damore, Kelley. "PCI bus catches favor among board makers." Computer Reseller News, p.87, p.89. May 16, 1994.

Davidson, Rob. "PCI bus takes on more factory work." Electronic Engineering Times, p.46, p.56. July 24, 1995.

Del Corso, D., Kirrmann, H., and Nicoud, J.D.. MICROCOMPUTER BUSES AND LINKS. Orlando: Academic Press Inc. (London) LTD 1986.

Evoy, Dave. "PC migration brings its share of headaches." Electronic Engineering Times Interactive, Issue 897, page 50. April 15, 1996.

Frank, Edward H., and Jim Lyle. SBus Specification B.0. U.S.A.: Sun Microsystems, Inc. 1990.

Goering, Richard. "Macros, cores trim ASIC design time." Electronic Engineering Times, p.77, p.80, p.83. November 6, 1995.

Kanellos, Michael. "ALL ABORD: PCI developors support decision -- APPLE SWITCHES BUSSES#." Computer Reseller News, p.79. December 19, 1996.

Klein, Bob, and Jyoti Basu. "ORCA FPGAs Move 155-Mb/s ATM Data to the PCI Bus." App Review, pp.12-13. June 12, 1995.

Lieberman, David. "PCI Spawns again." Electronic Engineering Times, p.73. March 27, 1995.

Raymond, Doug. "PCI juggernaut steals ISA slots." Electronic Engineering Times Interactive, Issue 897, page 28. April 15, 1996.

Runner, J. Scott, and Serge Rutman. "Reusable modules speed PCI design." Electronic Engineering Times, p.50. May 1, 1995.

Scheck, Susan. "PCI: It's Not Just For PCs Anymore." Electronic Buyer's News, p.31. March 13, 1995.

Shanley, Tom, and Don Anderson. ISA System Architecture. U.S.A. MindShare, Inc. 1991.

Shanley, Tom, and Don Anderson. PCI System Architecture, Second Edition. U.S.A. MindShare, Inc. 1993.

Solari, Edward. AT Bus Design. San Diego: Annabooks 1990.

Williams, Greg. "PCI--The Future of Macintosh Expansion." Apple Computer, Inc. Apple Directions, October 1995.

Wilson, Ron. "PLD vendors take on PCI." Electronic Engineering Times Interactive, Issue 810, page 46. August 15, 1994.

Wilson, Ron. "FPGA vendors revisit PCI" Electronic Engineering Times, p.86, p.88. January 22, 1996

Wilson, Tom. "PCI/68k bridge moves to the factory." Electronic Engineering Times, p.47, p.56. July 24, 1995.

Wirbel, Loring. "Multi-tiered PCI pushed for server I/O." Electronic Engineering Times, p.16. April 24, 1995.

Wittmann, Art. "Battle of the Buses- PCI Outperforms The Pack." Network Computing, pp. 140-141. February 1, 1995.

Wolfe, Alexander. "PCI: bus to future or dead end?." Electronic Engineering Times, p.1, p.130. April 1, 1996.

"Bus Mastering with the S5933 PCI Matchmaker", Application Note. Applied Micro Circuits Corporation, February 15, 1996.

EISA Specification Version 3.12. BCPR Services, Inc. 1989.

"Fully Compliant PCI Interface in an XC3164A-2 FPGA", Application Note. Xilinx, Inc., January 1995.

i960RP Microprocessor User's Manual, Intel Corporation, February 1996.

IEEE Standard for a High-Performance Synchronous 32-Bit Bus: MULTIBUS II. New York: The Institute of Electrical and Electronics Engineers, Inc, 1988.

IEEE Standard for a Versatile Backplane Bus: VMEbus. New York: The Institute of Electrical and Electronics Engineers. Inc. 1988.

MULTIBUS I Architecture Reference Book. Intel Corporation 1983.

PCI 9060 PCI Bus Master Interface Chip for Adapters and Embedded Systems. PLX Technology, Inc. 1995.

PCI Local Bus Specification Revision 2.1. PCI Special Interest Group 1992.

"PCI Local Bus A New ERA in Speed." Intel Technology Briefing from Internet at http://www.intel.com/product/tech-briefs/pcibus.htm. acquired on June 13, 1996.

Pentium and Pentium Pro Processor and Related Products. Intel Corporation, 1996.

Platform Components. Intel Corporation. 1997.

VxxxPBC User's Manual Local to PCI Bridge Controller. V3 Semiconductor 1996.

6. End Notes

[1] Solari, Edward. AT Bus Design, pp.173-174

[2]Solari, Edward. AT Bus Design, p. 2

[3] PCI Local Bus Specification Revision 2.1 p.25

[4] Shanley, Tom. ISA System Architecture, p.425

[5] Intel Pentium and Pentium Pro Processors and Related Products 1996, p.2-585

[6] Intel Platform Components 1997, p.1-32