U.S.N.A. – Trident Scholar project report; no.340 (2005)

A Network Interface Card for a Bidirectional Wavelength Division Multiplexed Fiber Optic Local Area Network

by

MIDN 1/C Joshua W. Wort, Class of 2005
United States Naval Academy
Annapolis, Maryland

Certification of Advisers' Approval

Associate Professor R. Brian Jenkins
Electrical Engineering Department

_____
(signature)

_____
(date)

CAPT Robert J. Voigt, USN
Electrical Engineering Department

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

_____
(signature)

_____
(date)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	5 May 2005	

4. TITLE AND SUBTITLE
A network interface card for a bidirectional wavelength division multiplexed fiber optic local area network

5. FUNDING NUMBERS

6. AUTHOR(S)
Wort, Joshua W. (Joshua Winston), 1982-

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
US Naval Academy Annapolis, MD 21402		Trident Scholar project report no. 340 (2005)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release; its distribution is UNLIMITED.	

13. ABSTRACT: In this project, a network interface card (NIC) for a fiber optic local area network has been designed and simulated. In the proposed network, each NIC has two optical transmitters and receivers on it, operating at different wavelengths. This allows the implementation of various network topologies on an optical fiber utilizing dense wavelength division multiplexing and bidirectional optical add / drop multiplexers. For example, a ShuffleNet network can be implemented on a single fiber optic ring. The ShuffleNet topology minimizes the number of hops through the network for a given data packet. For an eight node network, a maximum of three hops is required between any two of the nodes. In this network, control logic routes the data through the network by making decisions regarding transmitter selection. Because routing is performed on the NIC, the network control is more distributed than typical when using a central switch or hub. Distributed control, along with high data rate transmitters and receivers on each NIC, results in high network throughput and low latency. The predicted maximum aggregate throughput for this network is fifty gigabits per second, calculated by multiplying the number of transmitters (16) by the data rate of the transmitters (3.125 Gb/s).
.
.

14. SUBJECT TERMS: Network, Fiber Optic, Wavelength Division Multiplexing, Routing, ShuffleNet	15. NUMBER OF PAGES 102
	16. PRICE CODE

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

Abstract

In this project, a network interface card (NIC) for a fiber optic local area network has been designed and simulated. In the proposed network, each NIC has two optical transmitters and receivers on it, operating at different wavelengths. This allows the implementation of various network topologies on an optical fiber utilizing dense wavelength division multiplexing and bidirectional optical add / drop multiplexers. For example, a ShuffleNet network can be implemented on a single fiber optic ring. The ShuffleNet topology minimizes the number of hops through the network for a given data packet. For an eight node network, a maximum of three hops is required between any two of the nodes.

In this network, control logic routes the data through the network by making decisions regarding transmitter selection. Because routing is performed on the NIC, the network control is more distributed than typical when using a central switch or hub. Distributed control, along with high data rate transmitters and receivers on each NIC, results in high network throughput and low latency. The predicted maximum aggregate throughput for this network is fifty gigabits per second, calculated by multiplying the number of transmitters (16) by the data rate of the transmitters (3.125 Gb/s).

Keywords: Network, Fiber Optic, Wavelength Division Multiplexing, Routing, ShuffleNet

Acknowledgments:

My advisors, Assoc. Prof. R. Brian Jenkins and CAPT Robert Voigt, USN, for their eager assistance in every aspect of the project

Ms. Bonnie Jarrell, the Electrical Engineering Department Administrative Assistant, for all her dedicated work

The Electrical Engineering Department Lab Technicians, Ms. Daphi Jobe, Mr. William Stanton, and Mr. Gerry Ballman, for their expertise and assistance with facilities and equipment

ENS Adam Fisher, USN, U.S. Naval Academy Class of 2004, who did the Trident project which led up to mine

The Trident Scholar Committee for backing this project and Prof. Joyce Shade for her tireless efforts behind the scenes

My parents, Jonathan and Susie Wort, my sister Liz, my brother Daniel, and my girlfriend Valerie Ellingson for their constant support, encouragement, and prayers

My Lord and Savior for his sustaining grace and the peace that transcends all understanding which He provides in the most trying of times

Table of Contents

List of Figures

Background

Computer and communication networks are an integral part of everyday life, with applications in the office, school, and home, as well as on military bases and naval vessels. Networks allow users to share information and resources such as printers. There are many different ways to achieve interconnection in a computer network. Three of the most common topologies that exist today are the star, bus, and ring topologies shown in Fig. 1. In the bus topology that is used by the Ethernet standard, all the computers are connected to one common line and take turns transmitting. Because they share a line, only one node can transmit at a time. As users are added, service to individual nodes begins to diminish rapidly. In the star topology, each node is connected to a central hub or switch that manages point to point connections between nodes in addition to handling broadcast messages. This allows for better overall service for a large number of users, but does not allow one node to have more than one point to point link at a time. The ring topology has several different implementations. The most common is the token ring network. In a token ring, a message is passed around the ring that indicates that no one is transmitting. If a node wishes to transmit data, it takes this "network

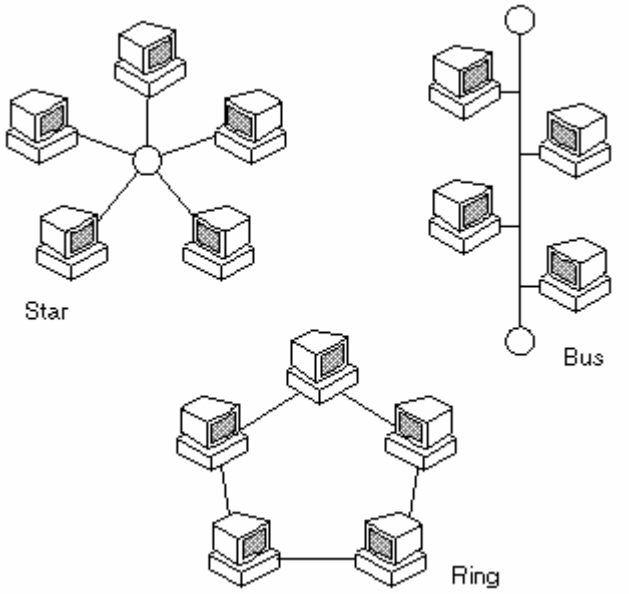


Fig. 1: Common Network Topologies[1]

free” message (the “token”) off the network and replaces it with the data packet that it wishes to transmit. When the packet makes it back to the sender, the sender removes its message from the network and places the token back into circulation. A token ring provides very good control of data transmission, but as the number of nodes on the ring increases, latency – the delay between when a packet is transmitted and when it is received - rises dramatically.

Each of these networks has a different protocol for controlling access by the users. Ethernet makes use of a protocol defined in the Institute of Electrical and Electronics Engineers (IEEE) standard 802.3, known as carrier sense multiple access with collision detection (CSMA/CD). This protocol resembles a polite dinner table conversation in which guests take turns talking. If two guests begin to speak at once, they pause, and one speaks while the other listens. The Ethernet protocol has no controlling mechanism that directs when each node transmits, and each transmission is broadcast to all other nodes. If the transmission is not intended for a given node, then that node will stop receiving the data, much like a dinner guest might tune out conversation that does not concern him.

The star network topology maintains more structured control of transmissions than traditional Ethernet. Star topologies often make use of a central switch that receives data from a user, reads the address information, and sends it to the designated recipient. Centralized control makes more efficient use of bandwidth, and eliminates the need for collision detection. Additionally, multiple pairs of nodes can have independent conversations simultaneously.

However, no matter what the topology of the network, the host at each node (very often a computer) needs some means to connect to it. The device used for this connection is known as a network interface card (NIC). In Ethernet, one of the most common types of local area networks (LANs), the NIC is the bridge between the computer and the network. It is the point where the

twisted pair data cable plugs into the back of the computer. It also contains the logic hardware that performs the CSMA/CD function of the Ethernet protocol, in addition to addressing and data encapsulation.

The IEEE has created a set of standards to govern the design of networks in order to ensure interoperability and compatibility. As part of these standards, IEEE created a model that describes the interaction of the different elements of a network, illustrated in Figure 2. The elements are described in a layered architecture where each layer provides some additional level of functionality.

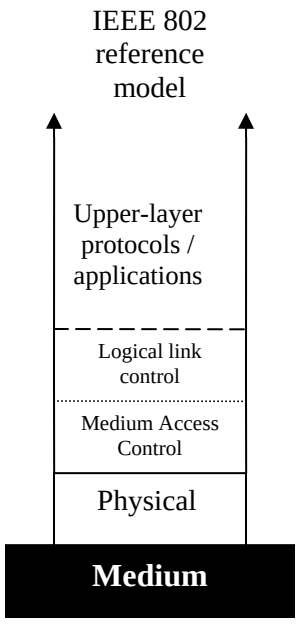


Figure 2: Network layer model[2]

At the very bottom of this layered model is the medium in which the data is transmitted, in the case of this project, optical fiber. Above that is the physical layer. This is the level at which the raw bit stream is transmitted and received. Above the physical layer is the data link layer, which consists of the medium access control (MAC) layer and logical link control. The MAC protocols handle network traffic, make routing decisions, and manage contention between nodes. The logical link control (LLC) layer is the interface between the upper layers where the user applications reside and the MAC layer.

Figure 3 shows the logical interconnection of any two nodes X and Y. As indicated by the dashed lines, the lower layers are “transparent” to the layers above. For example, the MAC layer of Node X behaves as if it were connected directly to the MAC layer of Node Y, as though the physical layers of each node were not there. This concept of a logical connection is easily

demonstrated using the example of a web browser, one type of application in the upper layers of the network stack. Communications between a web browser and an internet server are

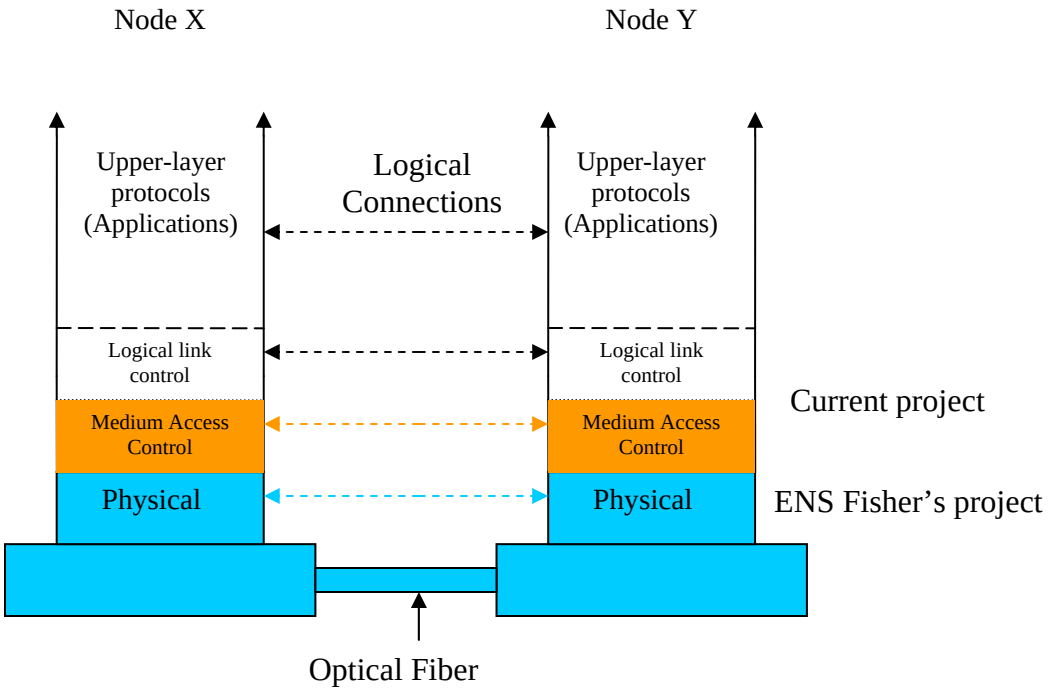


Figure 3: Interconnections of two nodes

independent of the lower layers, including the physical medium, which could be a dial-up connection, a wireless LAN, or a wired Ethernet.

The physical layer and the data transfer medium, shown in blue in Figure 3, was the emphasis of ENS Fisher's Trident Project last year. The current project focuses instead on designing and implementing the MAC layer. Since the MAC layer mediates access to the physical layer and must provide a reliable connection between two nodes, addressing, flow control, encapsulation, and error correction are important issues.

As a data packet moves down the stack of the layered model, pieces of information, known as headers, are appended to it. The effect is that at each lower layer, the information from above is encapsulated into a new protocol data unit. Each layer adds some information needed by

that layer, but most include some form of addressing. Figure 4 shows this process graphically, using Ethernet as the MAC layer. For this Trident project, the term "Ethernet header" in the diagram can be replaced with the term "MAC header". The other headers are from higher layer protocols such as Transmission Control Protocol (TCP) and Internet Protocol (IP) that operate beyond the scope of this project.

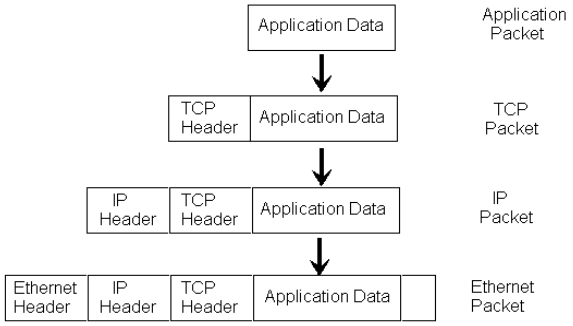


Figure 4: Data Encapsulation[3]

The MAC and LLC layers also manage the manner in which connections are established between hosts. In the case where hosts are not directly connected, then some type of switching must occur. There are several options. The first, known as circuit switching, is the most common protocol used in telecommunications. Any node that wishes to transmit sends out a call request signal. This signal is routed through the network, opening and reserving a circuit, or path, for transmission. The receiving node sends back a call accept signal. Upon receipt of the call accept signal, the transmitting node sends its data. After the data has been received, the receiving node sends an acknowledgment signal to the transmitting node and the connection is terminated. This protocol is illustrated in Figure 5a. Circuit switching is most efficient when sending large amounts of data at high speeds. However, the drawback is that it ties up the network resources between the two nodes so that during the setup, transmission, and breakdown times, those switches are unavailable for use by other nodes. For short bursts of data, typical of computer communications, this wastes bandwidth. To avoid this problem, networks often use a method of data transfer known as packet switching, illustrated in Figure 5c. In a packet switched network,

streams of data are divided up into packets of shorter lengths and transmitted through the network. The packets contain headers that describe their destination, and are routed accordingly. Packet switching is more bandwidth efficient since nodes and intermediate resources remain free

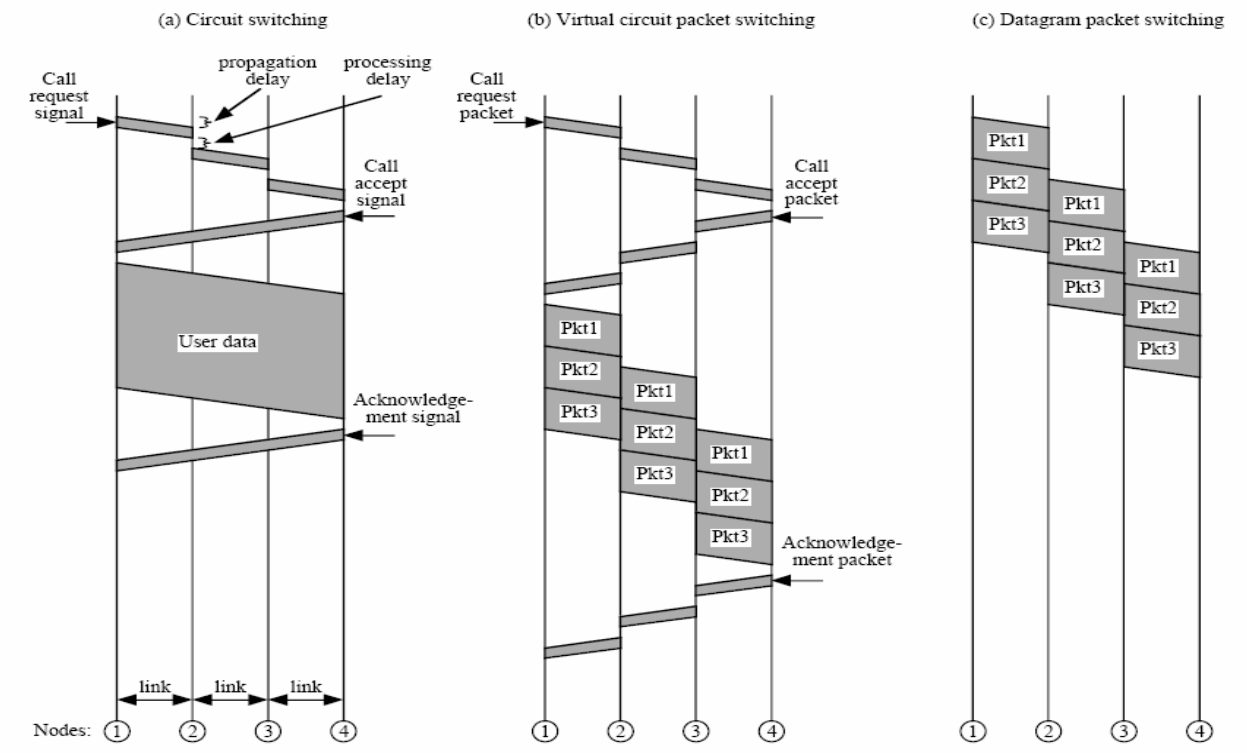


Figure 5: Switching Protocols[4]

for use, although sometimes a packet can be lost. In addition, the actual data transmission time may also be longer than that of a circuit switched network, so the latency can be larger. Another switching protocol that combines the benefits of circuit switching with those of packet switching is virtual circuit packet switching, illustrated in Figure 5b. In virtual circuit switching, the call request packet establishes the route for the data packets that will follow. However, instead of reserving the circuit for the sole use of the transmitting node, it leaves the nodes open for other transmissions. Often the data in the established circuit will have priority in the case of a packet collision, but during the gaps in transmission, other data is still free to flow.

This Trident project is concerned with the design of a MAC layer for new network topologies that use two transmitters and receivers on each NIC. One example is the topology implemented by ENS Adam Fisher in 2004 as a Trident project. In that project, a new means for implementing a ShuffleNet network was demonstrated. In an eight node ShuffleNet, as shown in Figure 6, each node has two inputs and two outputs, allowing an information packet to travel from one node to any other node in the network, making at most three hops in between. For instance, to send a message from Node 0 to Node 5 in the ShuffleNet shown in Figure 6, one option is to first transmit to Node 7 (first hop), then to Node 4 (second hop), then to Node 5 (third hop). Note that the four nodes on the left are repeated on the right to simplify the diagram. Such a configuration would most commonly be implemented using a dedicated line between each pair of nodes. Instead of using a separate line for each connection, ENS Fisher devised a way to interconnect all of the nodes with a single ring of optical fiber, as illustrated in Figure 7. Careful analysis of Figure 7 shows that the interconnections between the nodes are identical to those in Figure 6. To achieve the proper connectivity, light is transmitted in both directions on different wavelengths through the ring. One thing that it is important to note in Figure 6 is that each node has two transmitters and two receivers. This is not a common configuration, though it is currently used in the standard known as fiber distributed data interface (FDDI), a type of token ring network that uses optical fiber as

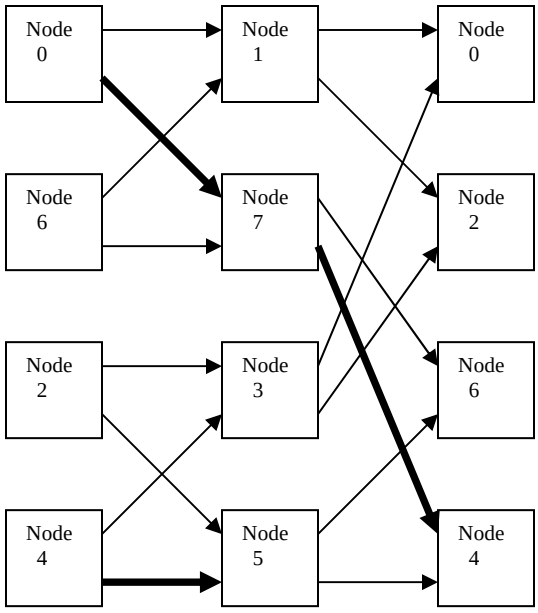


Figure 6: Spatial Representation of an eight-node network[5]

the data transmission medium. However, FDDI uses two rings that transmit in opposite directions (dual counter-rotating rings), and the two transmitters and receivers are used for redundancy rather than improved connectivity.

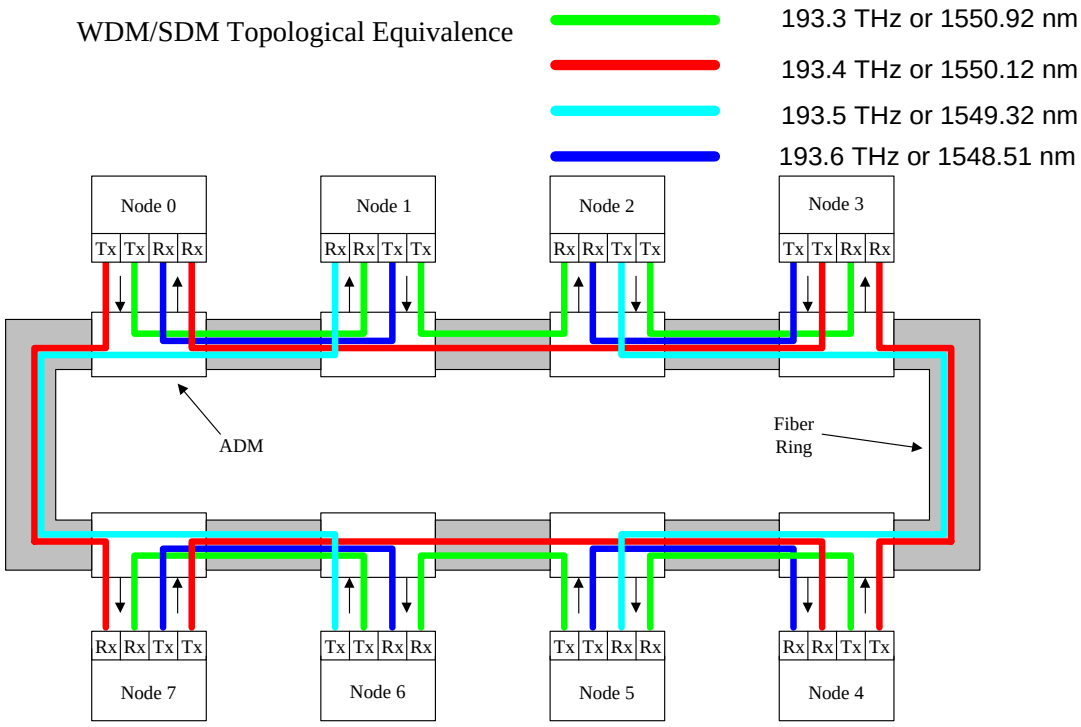


Figure 7: Eight-node network accomplished with four wavelengths and one fiber[5]

The MAC layer of this network topology has been designed for implementation on a NIC designed in this project. This NIC manages the two transmitters and two receivers, interfaces with the higher level layers in the protocol stack, and routes data packets. The control function of this network is much more distributed than that found in most networks. Ethernet NICs contain no routing control, a token ring's routing scheme is completely fixed (data flows in one direction until each node has handled it), and in a star network data goes through a central switch that makes all routing decisions made in the switch itself. This particular network configuration, with

the routing logic located on the individual NICs at each node, results in high aggregate network throughput.

Networks making use of optical fiber as a data transfer medium are have several significant advantages. First, optical fiber has a very large bandwidth, allowing high speed communication. Second, fiber is very lightweight and not subject to electromagnetic interference (EMI) to the same degree as traditional copper wired networks. As a result, it is very appealing for use in avionics applications where weight is at a premium. Research is currently being done to adapt optical fiber for avionics networks, particularly for the wide range of operating temperatures encountered by such networks. Varying temperature changes the dispersive properties of the fiber, making it more difficult to perform applications such as dense wavelength division multiplexing (DWDM) that are more sensitive to signal degradation. However, despite some of the difficulties, there is significant interest in the aviation industry for optical networks aboard aircraft. Such networks would need to support a broad range of data rates, ranging from a few kilobits per second up to several gigabits per second, depending upon the application making use of the network. On a commercial airliner, for instance, engine and environment sensors and flight controls (relatively low demand systems) would make use of the same network as the entertainment system (high demand)[10,11]. Military aircraft used for electronic warfare, electronic intelligence, or command and control could make particular use of such a high speed network because of the huge quantities of data handled onboard.

Other applications for this network include shipboard networks. Currently, data from sensors such as sonar must be pre-processed in order to avoid tying up the network. With the throughput promised by an optical ShuffleNet or similar network configuration, it would be possible to stream raw data to the central fire control computer, making more precise

calculations possible. Again, the light weight and lack of susceptibility to EMI is particularly appealing. The relatively compact size of optical fiber also would permit redundant lines in the case of physical damage to the primary network.

Design Overview

Figure 8 shows the overall concept of the design. This NIC is comprised of two transmitters and receivers (designated “Tx” and “Rx” respectively in the figure), a host interface

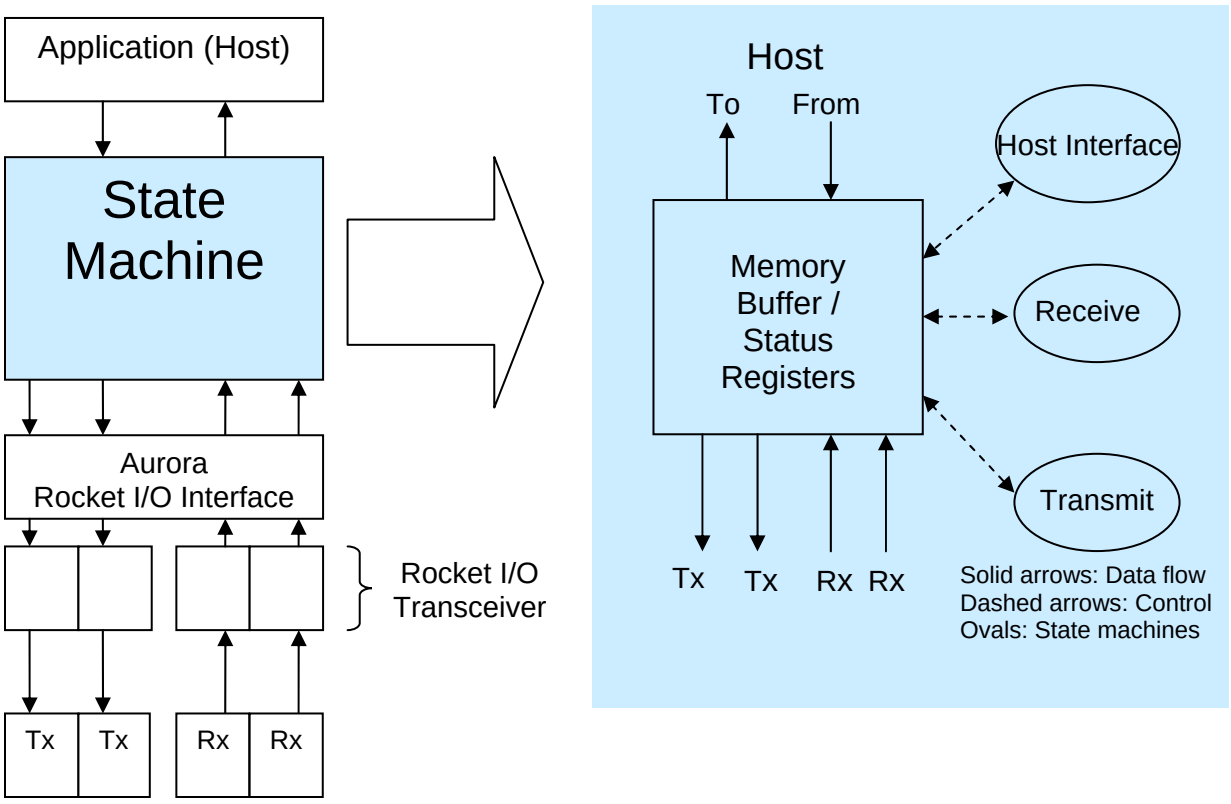


Figure 8: State Machine Interaction

(“To(T)” and “From(F)”), a block of memory, and three state machines (the ovals in the figure) to control data flow through the NIC. The receive state machine detects new data arriving on the receivers, selects a location for it in memory, directs it there, and notifies the transmitter or interface that a data packet has been received. The receive state machine is where the routing decisions take place. The transmitter state machine, when notified that a new packet is ready for transmission, accesses the memory and directs the data out the correct transmitter. The interface state machine performs both functions. It places data from the host in memory and notifies the transmit state machine, and it performs routing functions identical to that of the receive state

machine. It also receives memory addresses from the receive state machine and passes the data stored there to the host. As can be seen in the figure, data does not actually move through the state machines; rather, they are control devices for the memory, transmitters, and receivers.

The switching scheme that has been selected for this network most closely resembles virtual circuit (VC) switching. When a new data packet arrives on the receiver, the NIC reads the destination address of the packet and places it in a block of memory designated for the appropriate transmitter or the host interface. As can be seen in Figure 8, there are potentially three data sources for two transmitters. Packets can come from either receiver or the host. It is possible that the appropriate transmitter for a new packet may be busy. If the primary transmitter is available, the incoming data is streamed out the designated transmitter. If it is not, then the NIC places the data in a queue for that primary transmitter. If a ShuffleNet is used, as shown in Figure 6, every node eventually connects to every other. In such a network, it would be possible to send the packet out any available transmitter. This project, however, did not implement this design option, choosing instead to queue the data packets.

The determination of the appropriate transmitter is based on analysis of the incoming packet's MAC header, shown in Figure 9. The MAC header is three bytes long. The first byte (eight bits) is the destination address, the second byte is the source address, and the last byte is a priority field. Since this NIC is intended for the implementation of an eight node ShuffleNet, each

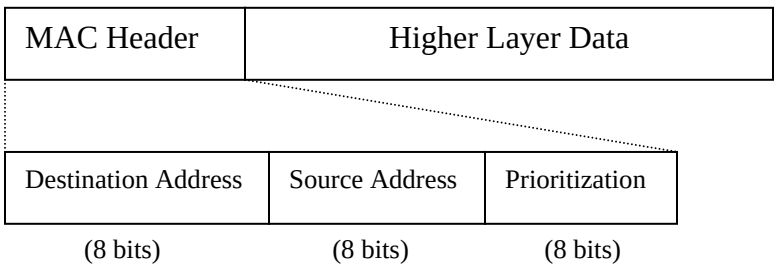


Figure 9: MAC Header Content

bit of the address field designates a node. Routing is accomplished by comparing the address to a stored value showing all the ideal destinations for a given transmitter, known as a mask. Based on that comparison, the data packet is either directed to the transmitter or placed in its queue.

This network is intended for a broad client base that could range from a normal office LAN to a network aboard a warship. As such, and particularly in the second role, it is desirable to have a prioritization scheme that will assign higher priority to certain data packets. In the case of a warship, this could be data from sensors or the fire control computer. As a result, there is a prioritization field in the packets passed to the LLC layer. Priority could be determined by the source address or the source service access point (SSAP) address. A shipboard network could conceivably use either, depending on the nature of the applications operating at priority nodes. If a fire control computer uses a standard TCP/IP interface with the LLC layer, it would be necessary to determine priority by examining the TCP socket designation. The TCP socket indicates the type of application for data coming down the stack and would be used to differentiate fire control data packets from e-mail data packets. If the workstation is dedicated solely to fire control and has no other applications available, which seems possible, it would be simpler to assign a priority level to the whole node so that all data leaving it is assigned a high priority. Due to time constraints on this project, the prioritization field has not been defined, though it is included in the header for use in future implementations.

The digital logic that makes up the switch was designed for implementation using a field programmable gate array (FPGA). An FPGA is an electronic chip that contains thousands of logic gates that can be programmed and, more importantly, reprogrammed from a computer. This gives a designer the ability to create multiple chips, test them, and fix any errors. In this project, the actual logic that drives the interface is implemented by three state machines that interact with

each other. A state machine represents the condition of a digital logic device. Each state defines a different situation, and transitions between states are driven by external stimuli or internal conditions, such as the expiration of a counter. A simple example for soda machine is illustrated in Figure 10. Most of the time, the machine is in an *Idle* state. When a coin or a dollar bill is inserted, it goes into a *Counting* state. Once enough money has been put in, it is in a *Ready to Vend* state. When the buyer makes a selection, it transitions to a *Vend* state, and once the soda has been dispensed, it returns to the *Idle* state. Each of these states has different actions associated with it (vend, count, etc.), and transitions between states are driven by measurable events (money inserted, sufficient funds, etc.). The state machines that control the NIC behave in the same manner. The receive state machine changes states based upon signals from the receivers. Signals from other state machines control its selection of a memory address. The transmit state machine responds to signals from its address queue, and its outputs drive the transmitters. The interface state machine responds to signals from both other state machines in addition to the interface with the host.

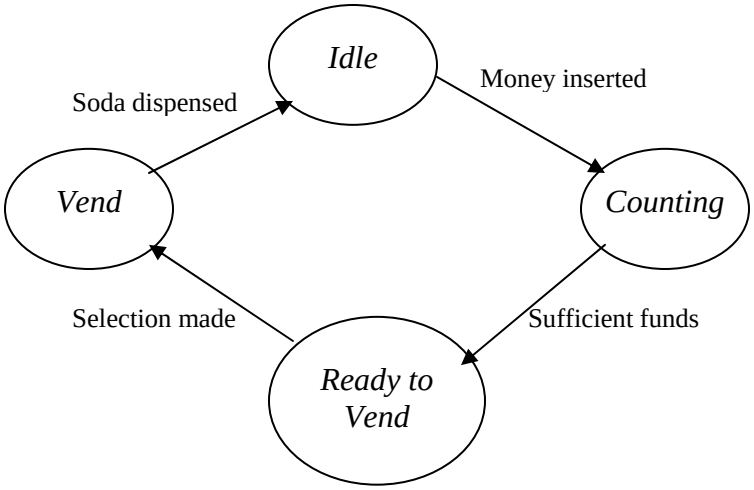


Figure 10: A Sample State Machine

Design Automation Environment

In order to program a state machine into an FPGA, it is necessary to enter the design using a software language that the chip can understand. One standard language for programming logic devices is Very High Speed Integrated Circuit (VHSIC) Hardware Definition Language (VHDL). The manufacturer of the FPGA being used in this project, Xilinx, provides an Integrated Synthesis Environment (ISE) which can be used to write and compile the code to be downloaded to the chip. This is the design entry step in the design process shown in Figure 11. Writing VHDL for a state machine can be a long and complicated process, with hundreds of lines of code required for state machines much simpler than the ones required for this project. Therefore, the ISE comes with a design utility known as StateCAD which allows the designer to enter a state diagram graphically. This is essentially a flowchart describing the nature of the states and the transitions between them, as was illustrated in Figure 10. From this state diagram, StateCAD generates the necessary VHDL code to implement the state machine. This saves considerable time, both in the actual entry of the code and in debugging (since StateCAD will not make the syntactical errors that a human coder is apt to make).

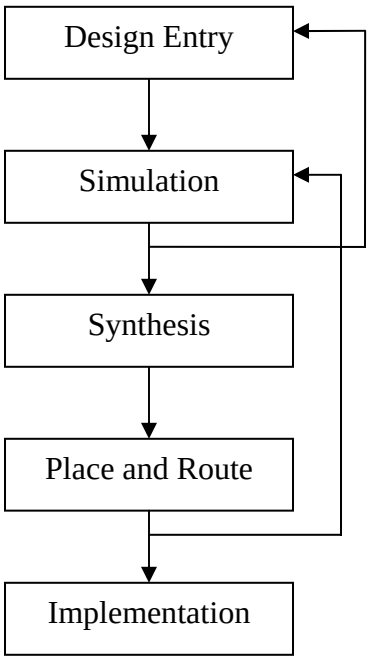


Figure 11: FPGA Design Process[6]

Once the VHDL has been generated, the ISE has several tools to examine the behavior of the code before it is downloaded to a device. StateCAD contains its own simulation software

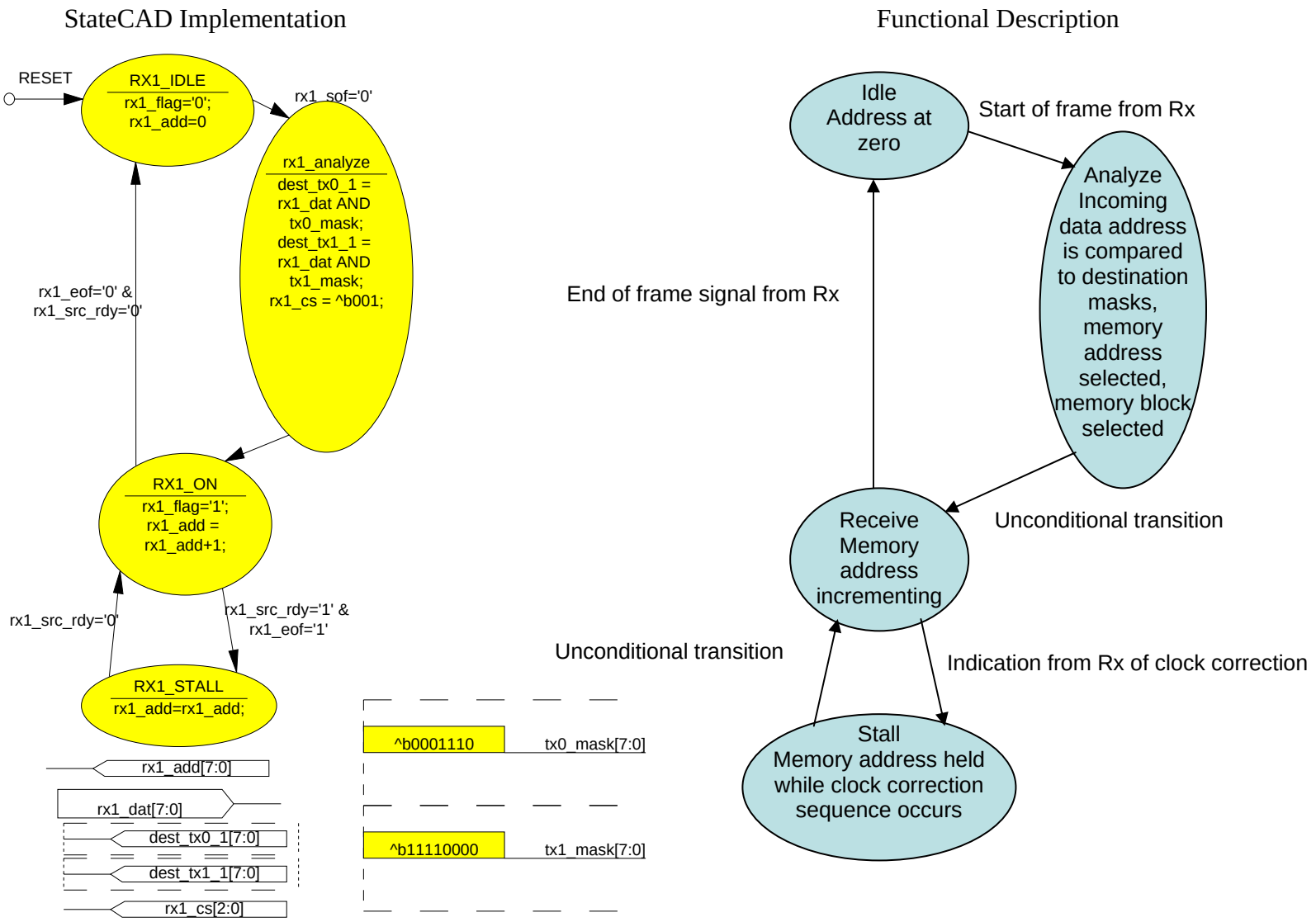
which allows a designer to test the response of the state machine to different conditions within the machine. Additionally, a simulation program known as ModelSim allows similar function with a design created in the ISE which contains more than just the state machine. In this project, the state machines were designed using StateCAD, and the VHDL was generated. That VHDL was loaded into the ISE where it was simulated using ModelSim. The state machines were designed and tested separately, then integrated together and tested as a unit. All simulations of the final versions of the state machines and the final NIC were conducted using ModelSim.

Once a functional design has been simulated, it is necessary to begin the process of translating it into hardware. This involves synthesizing the design, and placement and routing of the signals. During place and route, the physical location of each logic gate is determined. This is important because signal propagation through the hardware has a measurable effect on its performance. Poor placement could result in severely diminished performance due to propagation delay. Place and route affords the designer the opportunity to position elements of the design for optimum effectiveness, e.g. placing random access memory (RAM) blocks near the transceivers. Once the place and route process is complete, a file is generated by the ISE to program the device. This file is used to program the actual hardware connections in the FPGA that allow the parallel operation of all the devices in the NIC.

Design Details

The receive state machine, shown in Figure 12, performs the routing function of the NIC. When it detects a start of frame (SOF) signal from the receiver protocol, it captures the first byte of the packet (the destination address). The SOF signal causes the first transition of the receive state machine from an idle state to an analyze state. In this state, the state machine compares the destination address of the incoming packet to two "masks" stored in memory. This comparison shows whether the packet is destined for the host at that node, transmitter zero, or transmitter one (abbreviated Tx0 and Tx1 respectively). These masks are unique to each node, and can be changed without much difficulty in order to implement the different nodes of the network. Once the state machine has determined the destination of the packet, it sets the appropriate memory start address on the bus, selects an available block of memory, and begins storing the packet in the memory. This action places the identity of the memory block in the queue for the transmit state machine. The receive state machine continues to increment the address for the memory until it receives an end of frame (EOF) signal from the receiver, indicating that the whole packet has been received. Sometimes during the reception of data, the receiver will pause data transfer while it synchronizes with the transmitting device, a process known as clock correction. To accommodate this, the receive state machine contains a "stall" state. In this state, the address is held constant until the clock correction sequence is complete. The receive state machine actually is two state machines running in parallel, one for each receiver. In the figure, one has been replaced with a flowchart to better illustrate their function. However, StateCAD compiles the two into one VHDL file. When this file is converted to schematic form for placement in the design, it actually appears as one device.

Figure 12: Receiver State Machine State Diagram



The receive state machine is not implemented directly from the VHDL generated by StateCAD. Several transitions could be made more efficiently by using the StateCAD VHDL as a skeleton and editing some of the outputs by hand in the actual code generated. For example, the state diagram indicates a constant value for the chip select signals (“rx0_cs” and “rx1_cs”) which make the selection of the appropriate memory block. This is inaccurate; once the VHDL was generated with the appropriate output signals contained in the rx*n*_analyze state, the code was altered in order to simplify the chip select process. Additionally, memory address values were chosen based on the value of the dest_tx*n* signals contained in the state. This selection was also entered separately in the VHDL, which is appended in its entirety at the end of this report. Although this means of design was necessary for the receive state machine, the nature of the transmit state machine made it possible to implement it directly from the VHDL generated by StateCAD.

The majority of the design process focused on the state machines. However, the block random access memory (RAM) used to store the data has some unique features. In the schematic, three “chips” of block ram, each 192 bytes deep, are connected to the buses controlled by the state machines. Each chip has space for three sixty-four byte packets, as shown in Figure 13, and has two interface ports. One port is reserved for the receiver, and is configured to be a read / write (R/W) port. That is, data can either be written into memory by placing data on the input and incrementing the address port on the chip, or it can be read from the output port by inputting the desired memory address. The other port is for use by the transmitter, and as such, is a read only (R only) port. To retrieve data from memory, the transmit state machine places a valid memory address on the address port. The output is directly connected to the transmitter. All three chips in the memory contain data for all three possible destination sources (the two transmitters

and host attached to the node). This simplifies the process of writing data to the memory, because it guarantees that there will always be a memory block available for either receiver or the interface to write to. The receive or interface state machine performs the routing function by changing the address in the block RAM to which the incoming data packet is written. A packet destined for transmitter zero will be placed in the first 64 byte block on the selected chip, transmitter one in the second block, and so on. The identity of the selected chip is then placed in a queue maintained by the transmit state machine. Each transmitter has a fixed starting address; all that is required for a transmission is for the transmit state machine to select the appropriate memory chip and begin incrementing the address port. Figure 14 shows the interconnection of the memory with the rest of the NIC.

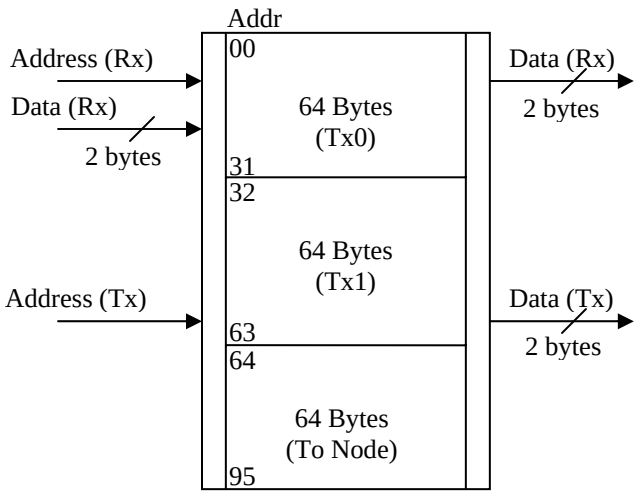


Figure 13: One Block RAM Chip

The arrows in Figure 14 most accurately represent data flow from the actual hardware receivers to the RAM and from the RAM to the transmitters. Each receiver (or the interface) has access to all three blocks of RAM through the switch. One of the outputs from the state machines is a chip select which makes the appropriate setups in the switch. Once a state machine has selected a block of RAM, the other state machines do not have access to it. The same is true on the transmit side of the memory.

The switches make use of tri-state buffers between the buses and the various potential drivers. The function of these signals is primarily to prevent collisions on the bus. If two separate entities try to simultaneously write different values to the same wire, the results can be catastrophic to the structure of the hardware. When a tri-state buffer is inactive, the output is essentially an open circuit. This disconnection from the bus prevents the catastrophically high currents associated with unequal simultaneous write operations. On the transmitter side, it was also necessary to implement this control logic on the chip select queues, as both receivers and the interface state machine would need access to all three queues. The tri-state buffers shown in Figure 15 are implemented on the receiver side. The chip select signals from the receiver are inverted within the device in order to properly drive the active-low tri-state buffers. The device contains two data routes: one that is two bytes wide for the data from the receiver, and one that is one byte wide for the address values from the receive state machine. The outputs from this device are tied directly to the RAM.

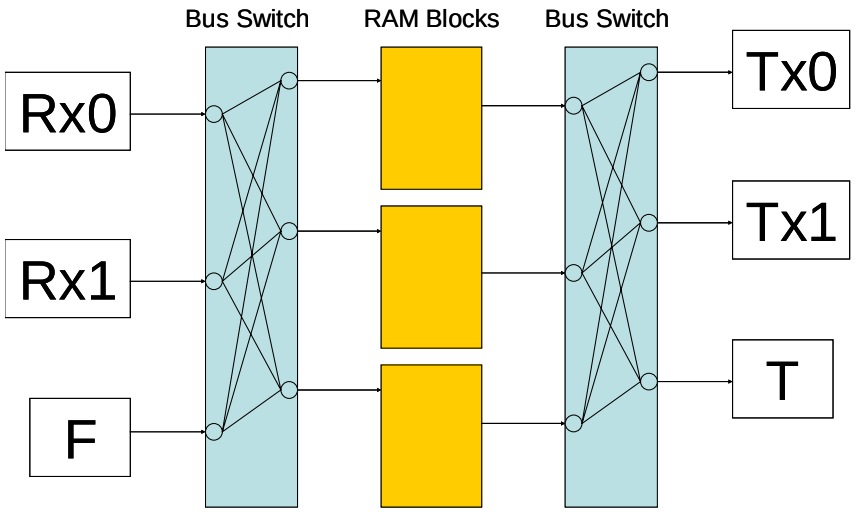


Figure 14: Memory Interface

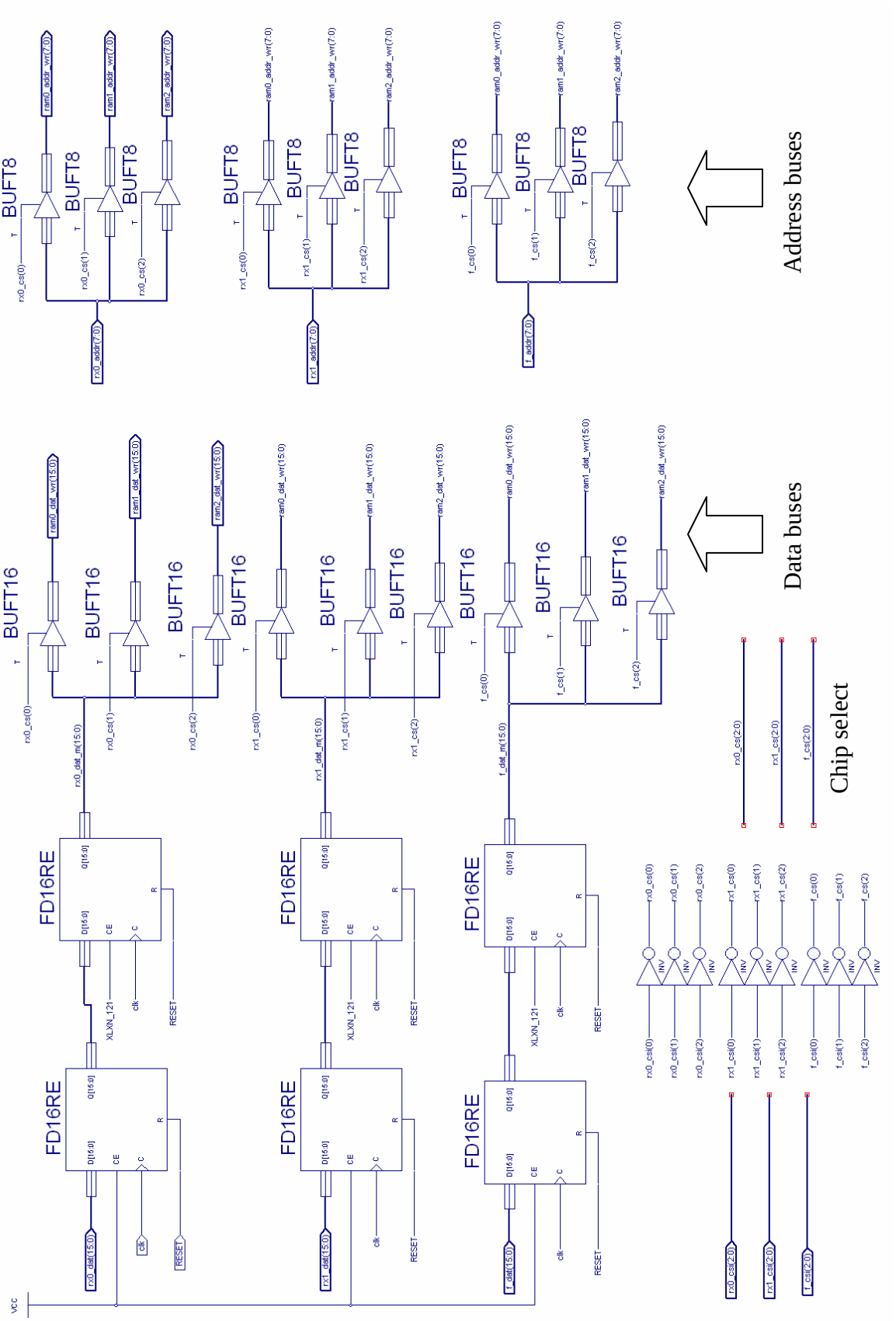
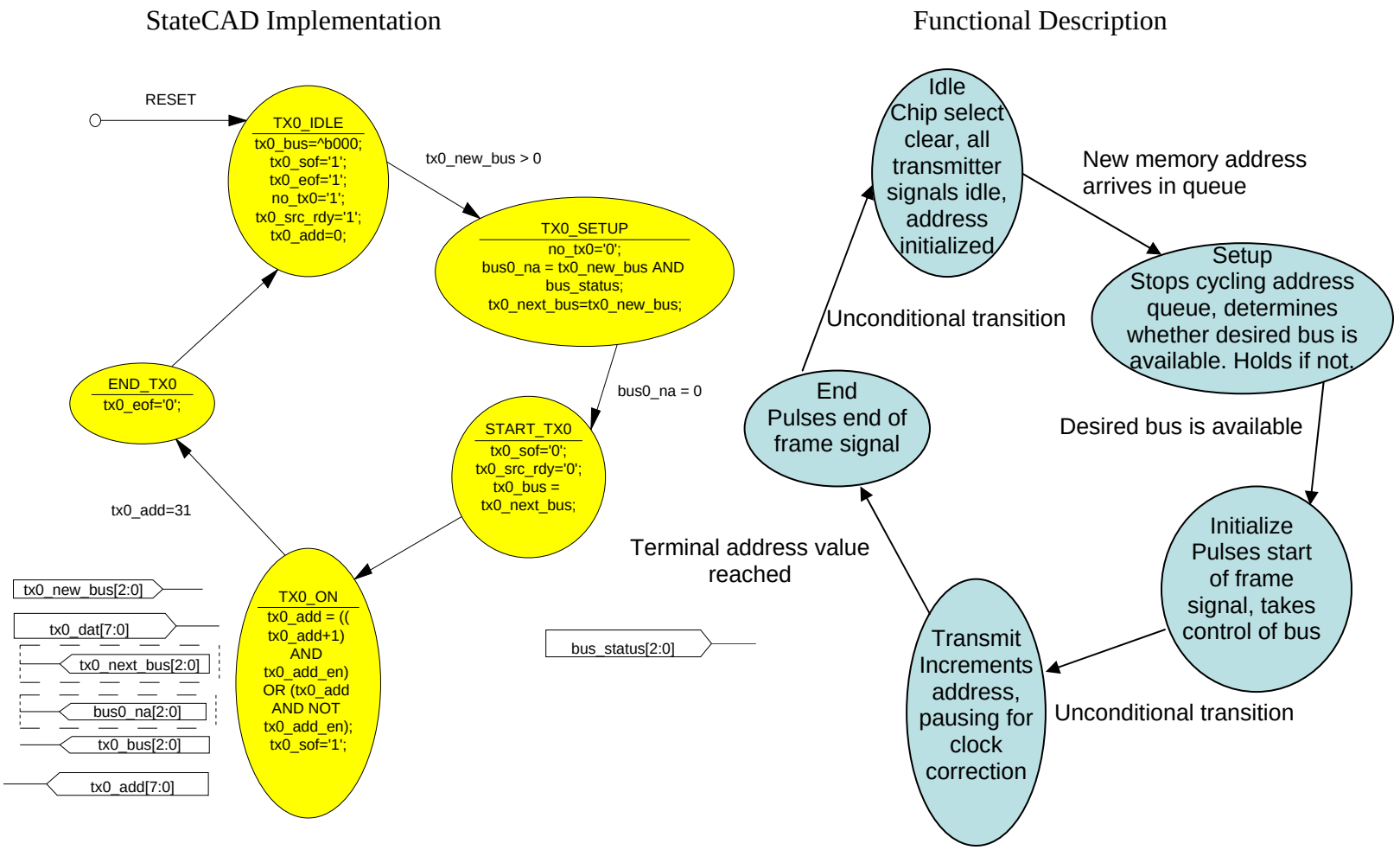


Figure 15: Chip Select Tri-state Buffers

The transmit state machine, shown in Figure 16, sits in an idle state until a new chip identity arrives at the top of its queue. A method of identity similar to the node addresses is used to identify the chip to the transmit state machine. The chip identity is three bits long, with one bit set high to indicate a given block of memory. As long as no new memory blocks have been written to, the bits in the queue are clear (zero). When a receiver or the interface starts writing to a block of memory, it places the identity of that chip in the transmit state machine's queue. The transition from idle to transmitter setup is driven by receipt of a memory identity greater than zero. The state following idle is a setup state (TX*N*_SETUP) in which the state machine prepares to access the appropriate chip. If the other transmitter or the interface is currently using that chip, the transmitter will wait in this "setup" state until the memory is available to read from (i.e. the bus becomes available) or a new packet comes in for the transmitter. Once the memory is available, the transmit state machine transitions to an intermediate state (START_TX*N*) in which it manipulates some signals that drive the transmitter module. It also connects to the memory at this point. It immediately transitions to its "transmitter on" state in which it drives the memory address port on the selected chip to retrieve the stored packet. Once it has retrieved the whole packet, it transitions to an intermediate state to manipulate the signals required for ending the transmission. One clock pulse later, it transitions to the idle state.

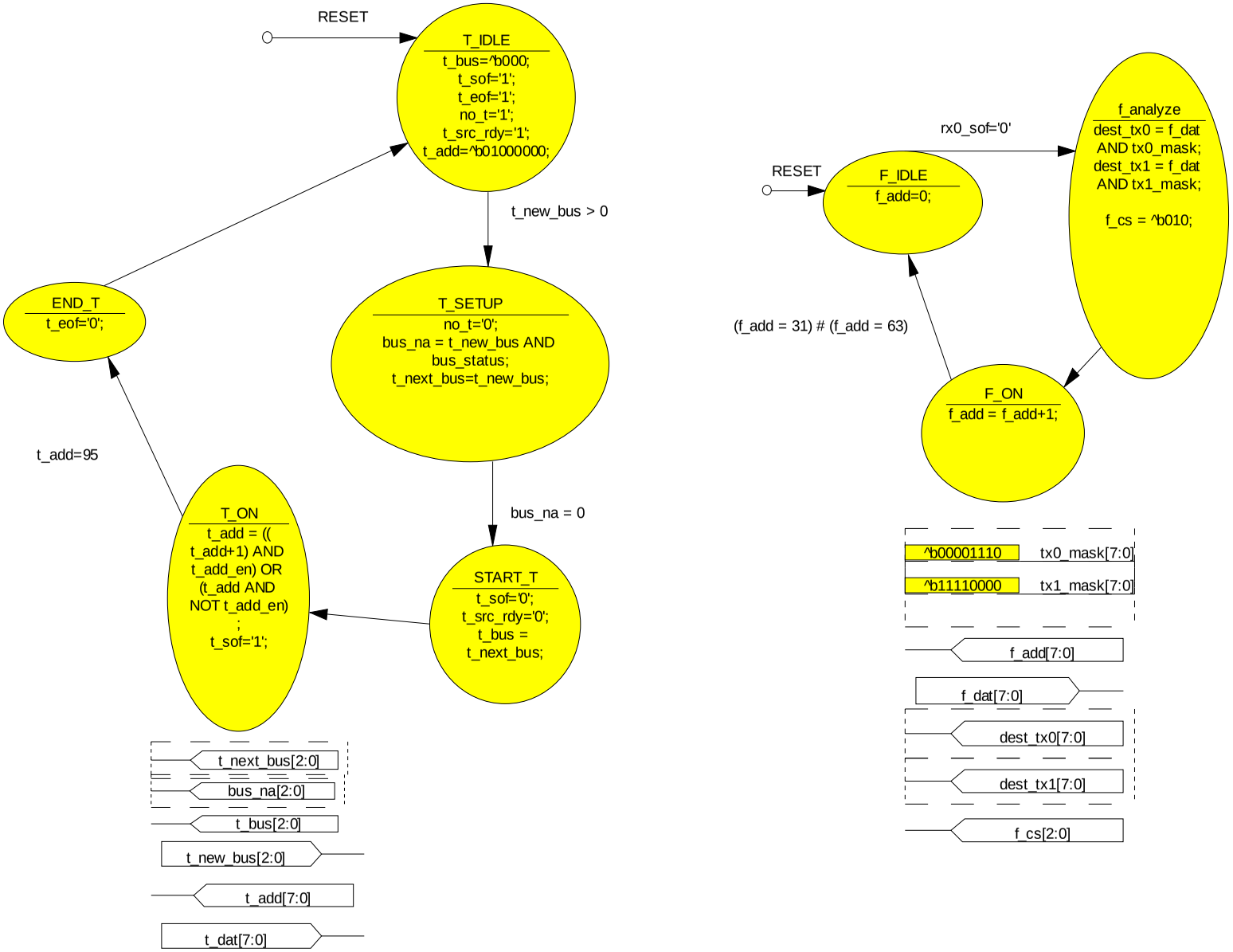
The transmit state machine is also two identical state machines operating in parallel, one for each transmitter. Like the receive state machine, it must be able to pause transmission if the transmitter initiates a clock correction sequence. It uses a gated counter to drive the address. The gate is tied to the signal on the transmitter that indicates if clock correction is taking place. If the signal indicates it is not, the counter increments. If the signal indicates that it is, then the counter holds its value until the gate opens again.

Figure 16: Transmit State Machine State Diagram



The interface state machine, shown in Figure 17, is also a pair of state machines. However, unlike the receive and transmit state machines, it must perform two different functions rather than perform one function on two different modules. Therefore, it contains elements of both other state machines. This implementation, with two separate state machines, assumes that the interface can transmit and receive simultaneously. If this were not the case, the interface machine could be modified with both processes returning to the same idle state rather than having the “To” and “From” state machines operating completely independently. Because there is no specification for the higher layer that the NIC will be interfacing with, the signals driving the interface state machine are more simplified than those of the other state machines. The interface still must contend with the other state machines for access to the memory, however, so many of those signals remain unchanged.

Figure 17: Interface State Machine State Diagram



Design Verification

The NIC was verified using the ModelSim simulator included in the ISE. The various state machines and components such as the tri-state bus switches were first tested separately to ensure proper operation. Once they had been verified, the overall design was assembled and simulated. Figure 18 shows the simulation of the transmit state machine. In this simulation, three packets are stored in memory in close order: one from each receiver and one from the node host(rx0_addr, rx1_addr, and f_addr respectively). Each is stored in a separate memory block, as can be seen by the respective chip select (cs) signals. The first packet to arrive, the one from the node, is stored in RAM2 (f_cs = 100), and is destined for transmitter zero, as indicated by the starting value of the address (zero). Three clock pulses later, transmitter zero begins to increment the read address on the correct chip. The delay is a result of the depth of the queue attached to the transmit state machine. Transmitter one completes its read without interruption. Meanwhile, another packet (rx0_addr) has been placed in the queue for transmitter zero. Once the first transmission is complete, the transmit state machine detects this packet and begins the cycle again. The packet for transmitter one (rx1_addr) is loaded without a problem. During the transmission, however, the transmitter pauses to perform a clock correction sequence, as shown by the not_tx1_dst_rdy signal pulsing low. During this time, the transmit state machine holds its address constant. Once the sequence is complete, it begins incrementing it again and continues through the end of the packet.

Figure 18: Transmit State Machine Simulation

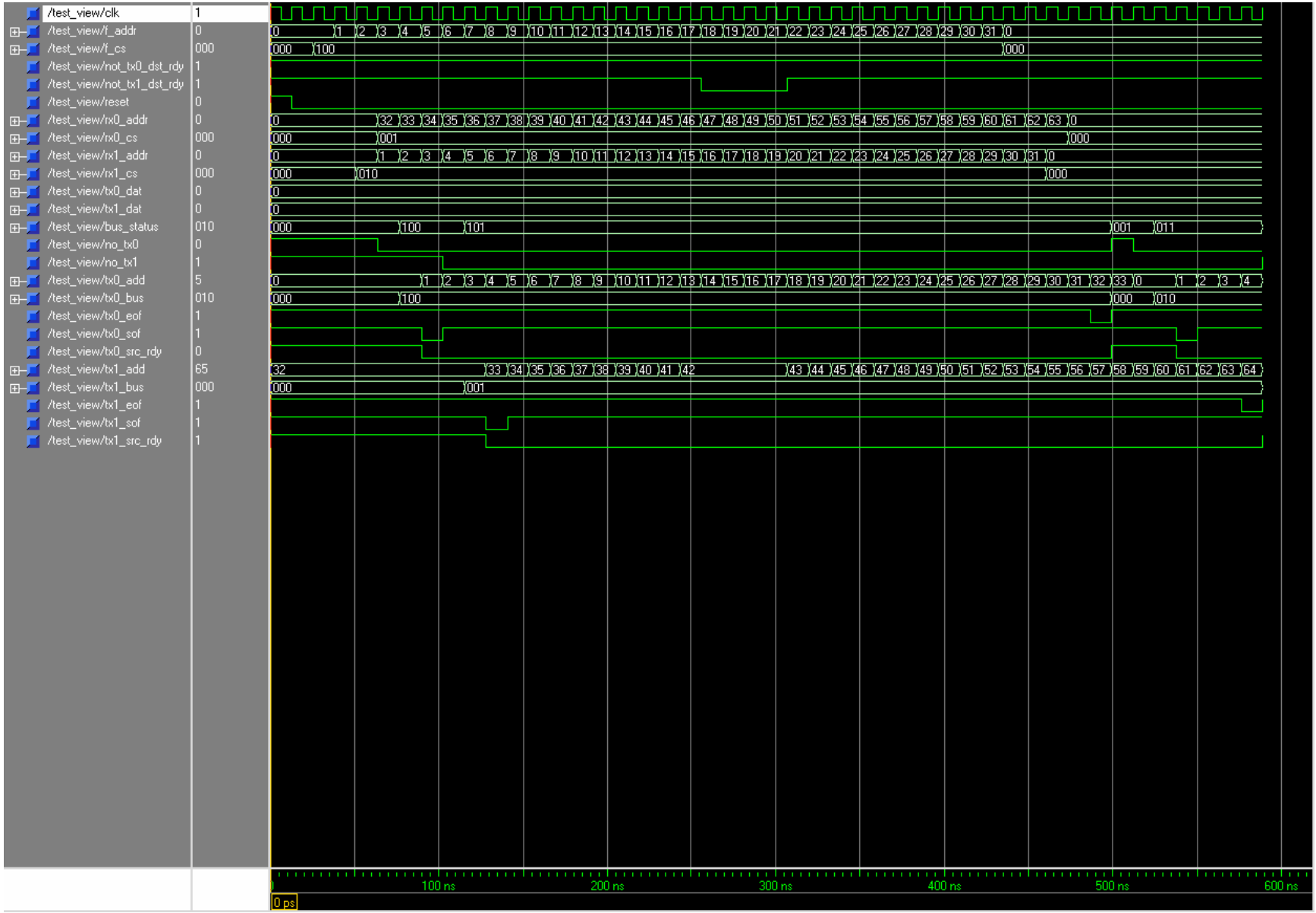


Figure 19 shows the simulation of the receive state machine. In this simulation, two different data packets are received. The first, coming in on receiver zero (rx0_dat), has a pause, as shown by the rx0_src_rdy signal pulsing high for a period during the reception. This packet is destined for transmitter one. The receive state machine handles this packet correctly, taking control of the first available bus (rx0_cs = 001), selecting the correct destination as shown by the beginning value of the address, incrementing the address until the pause, and continuing after the pause for clock correction has completed. The second packet, rx1_dat, is also destined for transmitter one. The receive state machine selects the first available bus (rx1_cs = 010), places the correct starting value for the address on the bus, and increments it until the end of frame signal is received.

Figure 20 shows the overall operation of the NIC. The highlighted signals show the path of a packet from where it comes in on receiver zero (rx0_dat) to its transmission out transmitter zero (tx0_dat). One clock pulse after it arrives on the receiver, the receive state machine begins storing it in the RAM (ram0_dat_wr). Lagging that signal by one clock pulse is the output from the read / write port of the RAM chip. This signal is only used to verify that the data has been properly stored in the memory. Six clock pulses after the initial write to the RAM, the transmit state machine begins to increment the address. Signal ram0_dat_r shows the data leaving the read port of the chip, and tx0_dat shows the same data going out the transmitter. The blue lines in the simulation window indicate that the tri-state buffers are operating as they should. The data, a string of characters, can be traced through the NIC and shown to move intact from receiver to transmitter. Another packet, destined for transmitter one, can be seen moving through the NIC a couple clock pulses after the initial test packet.

Figure 19: Receive State Machine Simulation

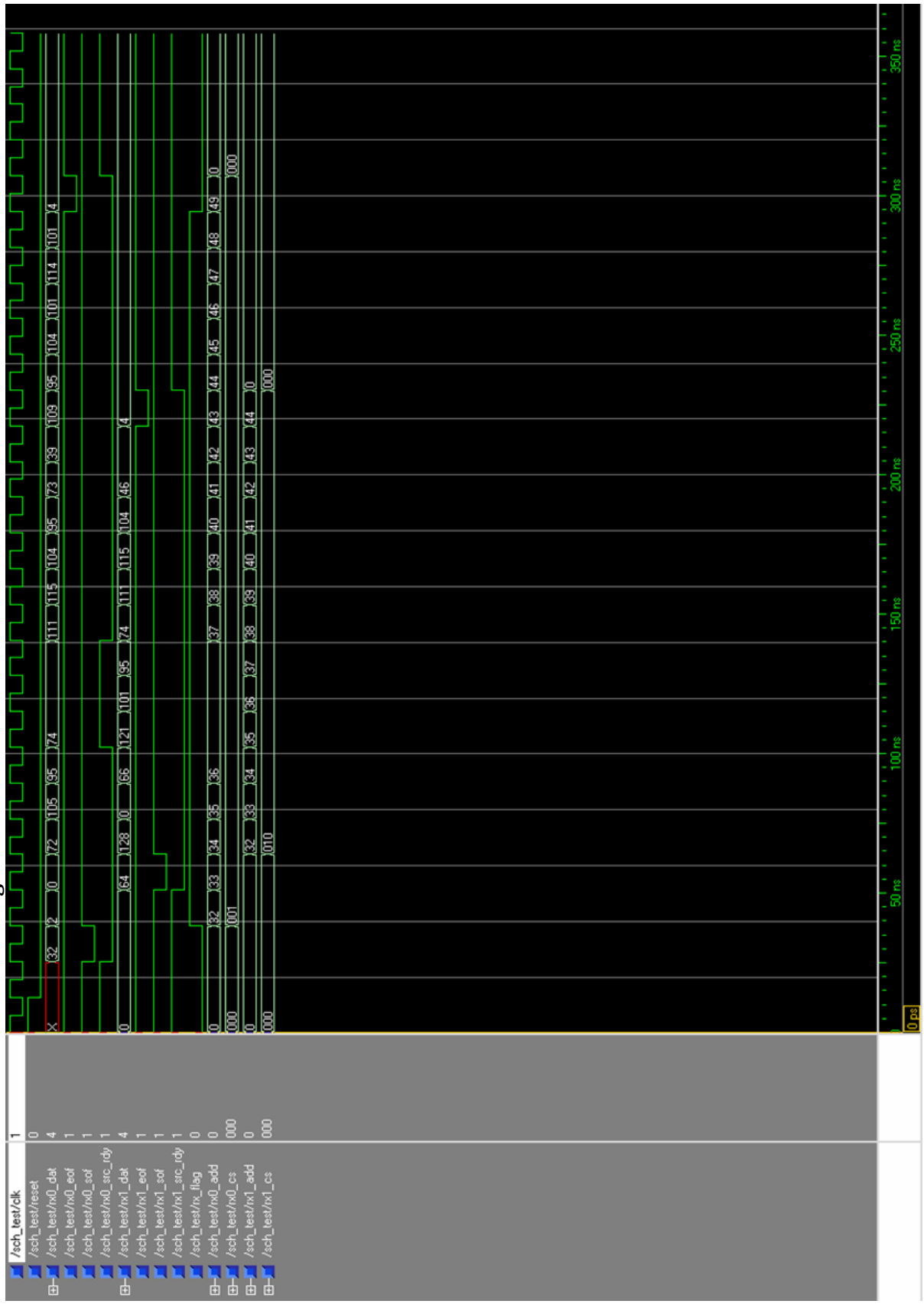
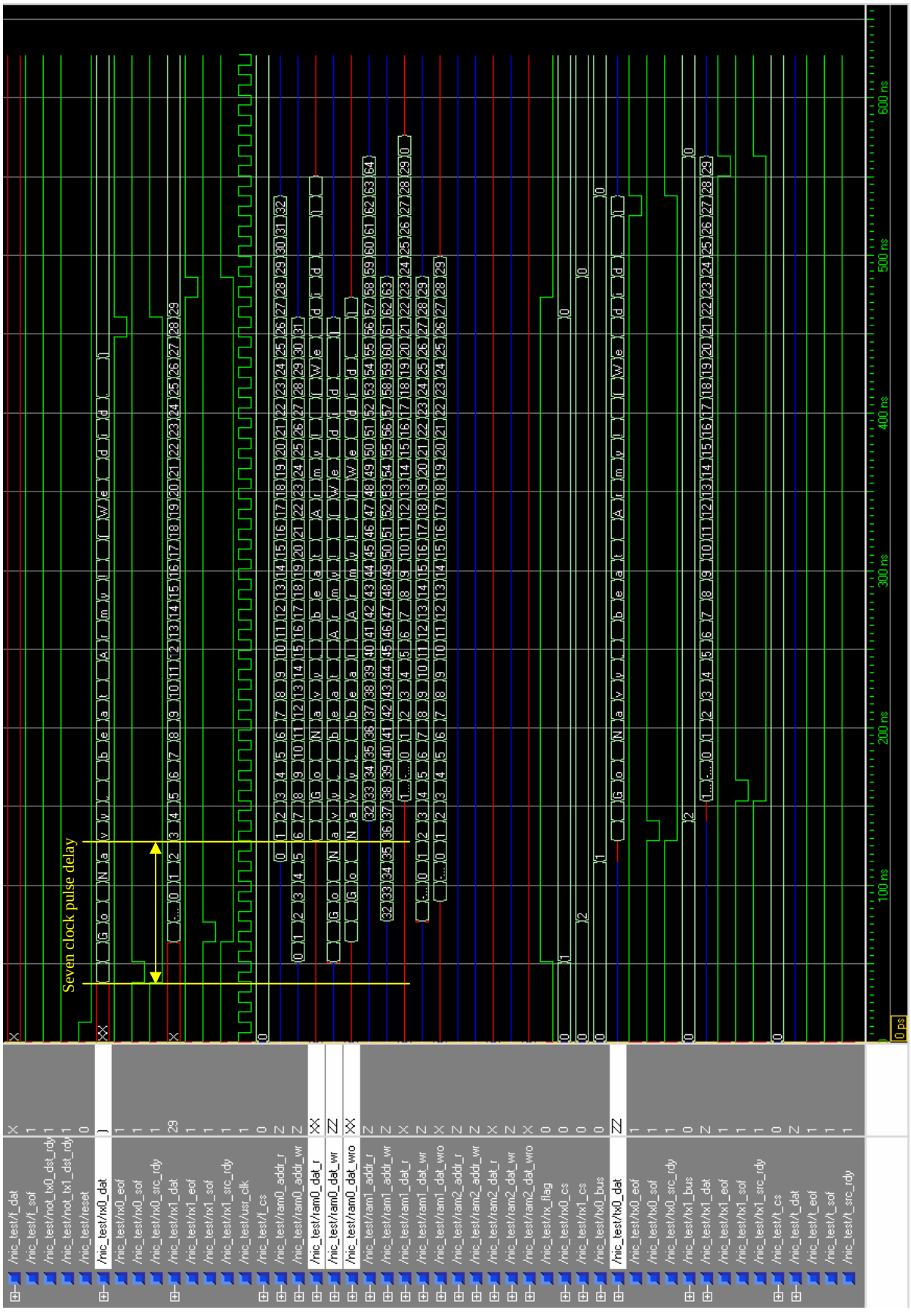


Figure 20: Overall NIC Simulation



Performance Analysis

Based on the clock speed necessary to achieve the desired data rate of 3.125 Gb/s, a clock rate of 156.25 MHz was selected. This corresponds to 6.4 ns for every sixteen bits of data. From the time the packet first arrives to the time transmission starts, there is a delay of seven clock pulses, or 44.8 ns total through the chip. The transceiver specification indicates a total delay of 42 clock pulses from the assertion of the start of frame signal on the transmitter to the assertion of the start of frame signal on the receiver at the other end. This is a latency of 268.8 ns, and it corresponds to the delay through the Aurora interface and the transceivers shown in Figure 8. Hence, the total latency through one NIC is 313.6 ns. The delay of 268.8 ns through the transceivers is approximately the same as the delay through fifty meters of optical fiber. For distances between nodes much greater than fifty meters, this delay is small compared to the optical delay.

The latency for the routing decision of seven cycles, or 44.8 ns, is very small, considering the service provided by the NIC. Ordinarily, to perform routing functions through a network, data must be passed to higher layer protocols. This often results in latency on the order of microseconds or even milliseconds – at least twenty times slower than the capabilities demonstrated by this NIC. At a clock rate of 156.25 MHz, the routing latency corresponds to a delay of 112 bits. This is 14 bytes, less than one fourth of the total packet length of 64 bytes. When the routing latency is smaller than the packet length, the routing process is fast enough to begin transmission before an entire packet has been received. This means that the switch control used in this design allows for one hundred percent utilization of the link bandwidth.

Design Implementation

The particular design board which was to be used for this project is the Virtex II Pro Development board produced by Avnet Electronics Marketing, shown in Figure 21. The board is intended for development of communications hardware and contains the Virtex II Pro FPGA manufactured by Xilinx. The board has several different features that made it appealing for this project. First, the FPGA contains the hardware for Xilinx's proprietary Rocket I/O transceivers which drive the small form-factor pluggable (SFP) modules and high speed serial data connector 2 (HSSDC2) ports. The Rocket I/O transceivers operate at data rates up to 3.125 Gbps. The SFP modules can be used to attach various devices, specifically two laser transceivers, to the board,

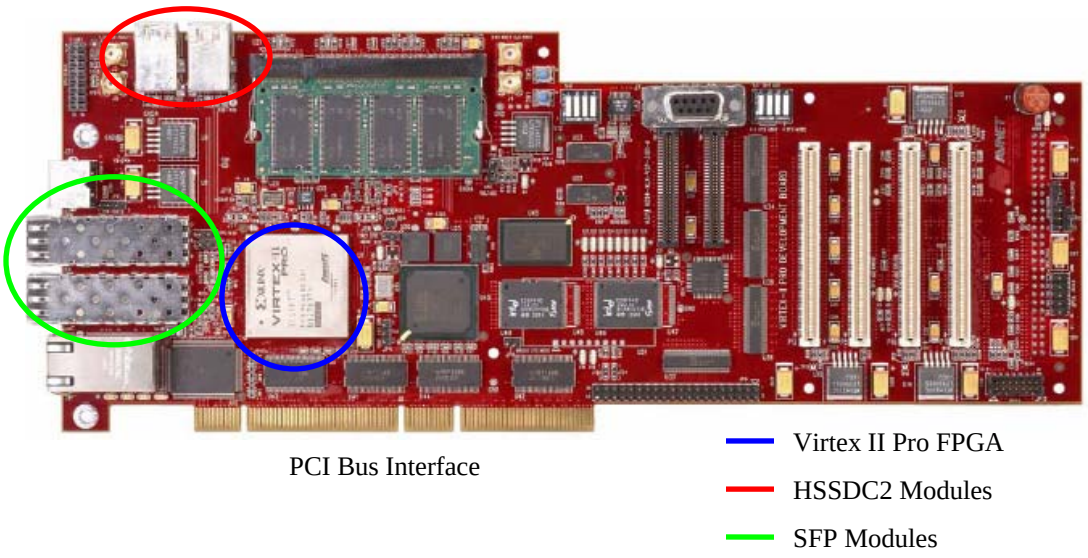


Figure 21: Virtex II Pro Development Board

while the HSSDC2 ports are connecting points for copper data cables. Additionally, the board has a PCI bus interface that allows it to plug into a computer internally. The PCI bridge hardware that comes with the board is intended to allow a designer to program the logic over the PCI bus as well as send data to and receive data from the hardware that is running on the board.

Another advantage of the development board was an intellectual property (IP) core known as Aurora. An IP core is a piece of proprietary software that performs some specific task. Aurora handles all the details of serial data transmission – channel encoding of data, channel initialization, and some basic error detection and correction. In the networking stack, Aurora is the physical layer. It takes the data that the user wishes to send, puts it in the proper format for transmission, and sends it across the link. In order to transmit the data over the link, Aurora takes the protocol data unit (PDU) supplied to it, in this case the MAC frame, and encodes it. Aurora uses a form of encoding known as 8B/10B encoding which transforms each eight bit byte into a ten bit word which is then transmitted. On the receiving end, it decodes the data and presents it to the MAC layer in the correct format for handling by the MAC layer

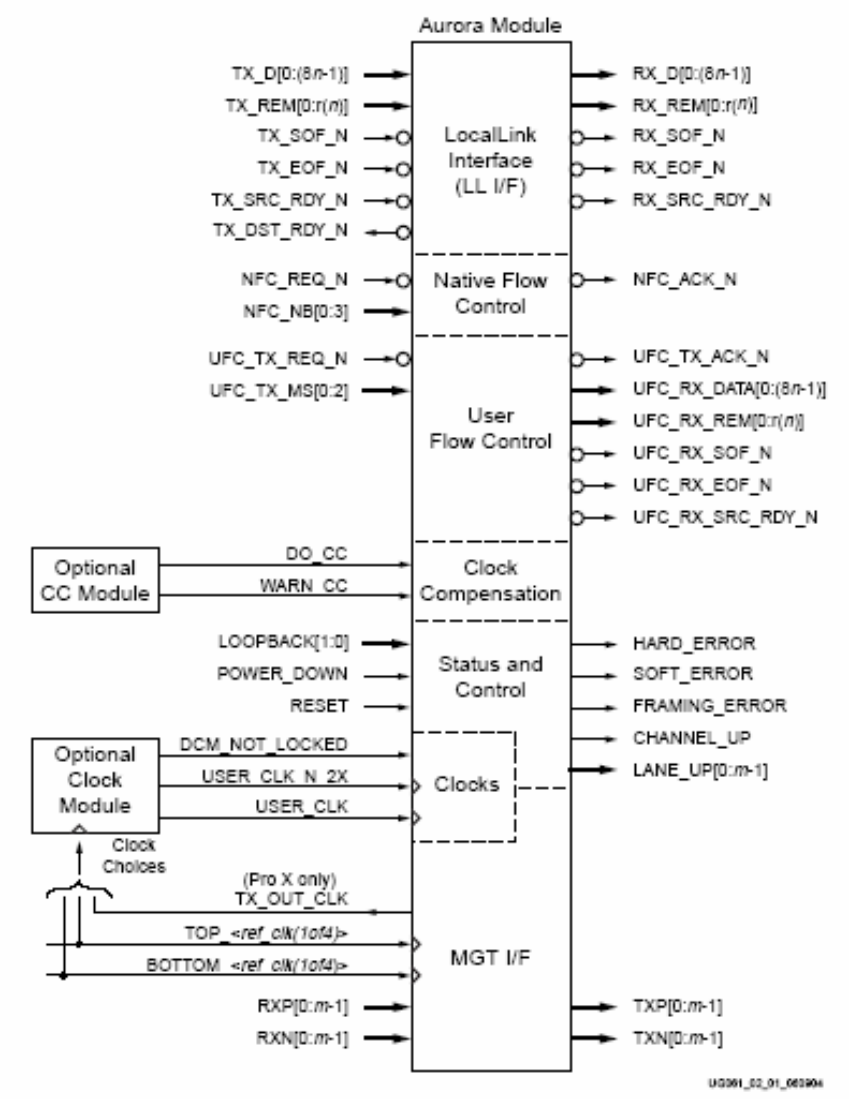


Figure 22: Aurora IP Core Signals[7]

protocols. This form of encoding allows for more efficient use of the transmitters and receivers at the physical layer, and it supports some error detection and correction.

Developments

Over the course of the first semester, there were several issues that delayed the design process. First, the supplier of the SFP laser transceiver modules which had been ordered to connect the board to ENS Fisher's fiber ring was unable to meet the delivery date that had been specified by their vendor. Apparently, the vendor had been misinformed as to the availability of the lasers. In lieu of those modules, it was decided to use other high speed serial ports on the board that could drive lasers already owned by the department. Unfortunately, the Aurora module could not be implemented in the design environment. For some reason, when the ISE tried to compile the code and synthesize the design, the process failed. It appeared to be missing a component lower in the design hierarchy, but the specified file was in the correct library. The same problem prevented simulation of the Aurora. However, the Aurora protocol specification provides some data that makes it possible to calculate the latency through the Aurora module.

The state machines were designed and implemented in the software and simulated using ModelSim. They functioned as expected, with one exception. The receive state machine transitions to idle before the final data word (two bytes) can be written to the RAM. This is due to the fact that the state transition is driven by the end of frame (EOF) signal asserted by Aurora simultaneous to the presentation of the last data word. There is a built in delay between presentation of the data to the state machine and presentation of the data to the memory to allow the state machine to perform its routing function. This delay causes a problem when the state machine transitions to idle at the conclusion of data reception, because the last word is not presented to the memory before the state machine closes the connection through the switch. This

problem could be very easily overcome by implementing a counter in the receiver state machine and linking the transition to it. However, this would prevent the state machine from being able to handle data packets of variable length, a desirable feature. Therefore, in the simulation, the end of frame signal is delayed one clock pulse in order to allow the state machine to store the full data packet.

Conclusion

Because of the many benefits afforded by the use of optical fiber in a communications network, further investigation of this subject is desirable. This project has provided a tentative first look at some of the options for operating a network with distributed control and two transmitters and receivers at the speeds required for fiber optic communications. The ShuffleNet configuration has many advantages, as mentioned earlier, but it is not the only option. Research into the adaptation of optical fiber for harsh environments such as avionics make it evident that there is interest in its use at the local area network level. Additionally, the products available for communications systems development, such as the development kit acquired for this project, demonstrate that there is a market for boards with two transceivers. Much of this work is likely to be proprietary, so there is limited research published on this topic.

This project has shown one possible implementation of control for such a network. Three state machines implemented on an FPGA are able to perform the necessary routing functions. Because of the fixed nature of the network, it is possible to use a mask at each node which makes the routing decision in one clock pulse. This allows for quick retransmission of the data if there is a transmitter available. Additionally, the use of dual port block RAM permits the transmitter to begin sending a data packet before it has been completely stored in the memory, further reducing

the latency through the device. Using three chips that are switched to the individual transmitters and receivers guarantees availability of memory for writing.

Further developments on this project could include the implementation of the higher layer protocols to connect the interface state machine to an actual host. This project is an intermediate step in the use of fiber optic communications in local area networks. The low latency demonstrated by this NIC represents significant progress in network routing. This latency, due to the conversion between optical and electronic signals, is typically the largest contributor to latency in any optical network. MIDN Jessop has proposed a Trident project for next year which will explore the possibility of performing switching and routing operations by doing wavelength conversion. This would eliminate the need to convert to an electronic signal unless it is necessary to pass it to the host at that node. However, wavelength conversion is an emerging technology. This project has shown that a high speed optical ShuffleNet is feasible using existing technology.

Endnotes

1. http://networking.webopedia.com/FIG/TOPOLOGY.gif

2. William Stallings, *Business Data Communications, 4th ed.* Upper Saddle River, New Jersey: Prentice Hall, 2001.

3. http://www.comptechdoc.org/independent/networking/guide/dataencapsulation.gif

4. William Stallings, *Data & Computer Communications, Sixth Edition*. Upper Saddle River, New Jersey: Prentice Hall, 2000.

5. Adam Fisher, "A Wavelength Multiplexed Bidirectional Fiber Ring Network". USNA Trident Project Interim Report, 2003.

6. http://www.netrino.com/Articles/ProgrammableLogic/

7. Aurora Reference Design User Guide, UG061 (v2.2). Xilinx Corporation, October 26, 2004.

Bibliography

Aurora Reference Design User Guide, UG061 (v2.2). Xilinx Corporation, October 26, 2004.

Chan, Pak K., and Samiha Mourad, *Digital Design using Field Programmable Gate Arrays*. Englewood Cliffs, NJ: Prentice Hall, 1994.

Chang, K.C., *Digital Systems Design with VHDL and Synthesis: An Integrated Approach*. Los Alamitos, California: IEEE Computer Society, 1999.

http://www.comptechdoc.org/independent/networking/guide/dataencapsulation.gif

http://www.netrino.com/Articles/ProgrammableLogic/

http://networking.webopedia.com/FIG/TOPOLOGY.gif

Fisher, Adam. "A Wavelength Multiplexed Bidirectional Fiber Ring Network". USNA Trident Project Interim Report, 2003.

Gerla, Mario, et al., "Routing in the Bidirectional Shufflenet," *IEEE/ACM Transactions on Networking* Vol. 9, no.1 (February 2001): 91.

Glista, Andrew S., Jr., and Mark W. Beranek, "Wavelength Division Multiplexed (WDM) Optical Technology Solutions for Next Generation Aerospace Networks," IEEE/AIAA Digital Avionics Systems Conference Proceedings (22nd DASC), Oct. 2003.

Karol, Mark J., and Richard D. Gitlin, "High-Performance Optical Local and Metropolitan Area Networks: Enhancements of FDDI and IEEE 802.6 DQDB," *IEEE Journal on Selected Areas in Communications*, Vol. 8, no.8 (October 1990): 1439.

Kleitz, William, *Digital Electronics: A Practical Approach, Sixth Edition*. Upper Saddle River, New Jersey: Prentice Hall, 2002.

McLaughlin, Andrew J., et al., "Design and Development of a Multi-Gigabit WDM Network for use in the Aerospace Environment," 2000 IEEE Aerospace Conference Proceedings, Volume 3, 18-25 March 2000 Pages: 115-124 vol. 3.

Stallings, William. *Business Data Communications, 4th ed.* Upper Saddle River, New Jersey: Prentice Hall, 2001.

Stallings, William. *Data & Computer Communications, Sixth Edition*. Upper Saddle River, New Jersey: Prentice Hall, 2000.

Appendix A: Glossary

Aurora: A piece of proprietary software owned by Xilinx Corporation and provided to designers for use in designing high speed serial data communication links.
Avionics: Aviation electronics; electronic devices designed for use on board aircraft.
Buffer: A memory location where data is temporarily held.
Bus: A wire or collection of wires held in common by several different entities.
CSMA/CD: Carrier Sense Multiple Access / Collision Detection, the protocol used by Ethernet to arbitrate use of the data transfer medium.
Datagram: A data packet sent through a network that contains part or all of a data stream (depending on the size of the packet) as well as addressing information.
Encapsulation: The process of breaking up a stream of data into packets that can be transmitted through a network and which contain headers with addressing information to be used at each layer of the network stack. See Figure 4, Page 7.
Ethernet: A common LAN standard, described in the IEEE 802.3 standard.
FDDI: Fiber Distributed Data Interface, a network topology that uses two rings of optical fiber that transmit in opposite directions. One of the rings is for redundancy.
FPGA: Field Programmable Gate Array, a reprogrammable hardware chip that allows for the design, testing, and troubleshooting of logic circuits.
Framing: The encapsulation of data for transmission at the bottom of the networking stack.
Header: Data bits on the beginning of a datagram that contain information such as source and destination address and prioritization.
Host: The device connected to the network at a node.
HSSDC2: High Speed Serial Data Connector 2, a hardware interface that uses copper wire for serial data communication.
IEEE: Institute of Electrical and Electronics Engineers, an international organization that oversees standards in electrical and computer engineering.
LAN: Local Area Network, an interconnection of computers and other devices whose physical scale usually does not exceed that of an office building.
LLC: Logical Link Control, the sublayer above the MAC sublayer in the IEEE model which interfaces with applications in higher layers of the host.
MAC: Media Access Control, a sublayer of the IEEE networking protocol model in which controls for transmitting and receiving data as well as arbitrating use of transmitters reside.
NIC: Network Interface Card, a piece of hardware that allows a device to connect to and use a network.
Node: A connection point to a network, usually referring to the device connected.
PCI Bus: Peripheral Component Interconnect, an interface used to plug a device into the motherboard of a computer. Used for such devices as sound cards, video cards, modems, and NICs.
PDU: Protocol Data Unit, a data packet.
Rocket I/O: A proprietary driver contained in the Xilinx Virtex II Pro chip for use with high speed serial data communication (supports data rates up to 3.125 Gb/s).
Serial: Data transmission where bits are sent sequentially over one line.
SFP: Small Form-factor Pluggable, a hardware interface for the connection of various devices. This project intended to use laser transceivers that would connect to the SFP interfaces on the design board.

State machine: A means of describing electronic hardware as several sets of conditions, known as states. Transitions between states are caused by timers or other signals.
TCP/IP: Transmission Control Protocol / Internet Protocol, two protocols that reside above the LLC layer of the IEEE networking model.
Transceiver: A device that contains both a transmitter and a receiver.
VHDL: Very High Speed Integrated Circuit (VHSIC) Hardware Definition Language, a programming language used to describe logic circuits.

Appendix B: State Machine VHDL Code

Receive State Machine:

```
-- C:\XILINX\TRIDENT\RECVTEST\RECV.vhd
-- VHDL code created by Xilinx's StateCAD 6.2i
-- Sat Apr 09 15:59:32 2005

-- This VHDL code (for use with IEEE compliant tools) was generated using:
-- enumerated state assignment with structured code format.
-- Minimization is enabled,  implied else is enabled,
-- and outputs are speed optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY SHELL_RECV IS
        PORT (CLK,RESET,rx0_dat0,rx0_dat1,rx0_dat2,rx0_dat3,rx0_dat4,rx0_dat5,
                rx0_dat6,rx0_dat7,rx0_eof,rx0_sof,rx0_src_rdy,rx1_dat0,rx1_dat1,rx1_dat2,
                rx1_dat3,rx1_dat4,rx1_dat5,rx1_dat6,rx1_dat7,rx1_eof,rx1_sof,rx1_src_rdy,
                bus_0_busy,bus_1_busy,bus_2_busy: IN std_logic;
                rx0_add0,rx0_add1,rx0_add2,rx0_add3,rx0_add4,rx0_add5,rx0_add6,rx0_add7,

        rx0_cs0,rx0_cs1,rx0_cs2,rx1_add0,rx1_add1,rx1_add2,rx1_add3,rx1_add4,rx1_add5
                ,rx1_add6,rx1_add7,rx1_cs0,rx1_cs1,rx1_cs2,rx_flag : OUT std_logic);

        SIGNAL BP_rx0_add0,BP_rx0_add1,BP_rx0_add2,BP_rx0_add3,BP_rx0_add4,
                BP_rx0_add5,BP_rx0_add6,BP_rx0_add7,BP_rx0_cs0,BP_rx0_cs1,BP_rx0_cs2,

        BP_rx1_add0,BP_rx1_add1,BP_rx1_add2,BP_rx1_add3,BP_rx1_add4,BP_rx1_add5,
                BP_rx1_add6,BP_rx1_add7,BP_rx1_cs0,BP_rx1_cs1,BP_rx1_cs2,dest_tx00,
                dest_tx0_10,dest_tx0_11,dest_tx0_12,dest_tx0_13,dest_tx0_14,dest_tx0_15,
                dest_tx0_16,dest_tx0_17,dest_tx01,dest_tx1_10,dest_tx1_11,dest_tx1_12,
                dest_tx1_13,dest_tx1_14,dest_tx1_15,dest_tx1_16,dest_tx1_17,dest_tx02,
                dest_tx03,dest_tx04,dest_tx05,dest_tx06,dest_tx07,dest_tx10,dest_tx11,
                dest_tx12,dest_tx13,dest_tx14,dest_tx15,dest_tx16,dest_tx17,rx0_flag,rx1_flag
                ,tx0_mask0,tx0_mask1,tx0_mask2,tx0_mask3,tx0_mask4,tx0_mask5,tx0_mask6,
                tx0_mask7,tx1_mask0,tx1_mask1,tx1_mask2,tx1_mask3,tx1_mask4,tx1_mask5,
                tx1_mask6,tx1_mask7: std_logic;
END;

ARCHITECTURE BEHAVIOR OF SHELL_RECV IS
        TYPE type_sreg IS (rx0_analyze,RX0_IDLE,RX0_ON,RX0_STALL);
```

```
        SIGNAL sreg, next_sreg : type_sreg;
        TYPE type_sreg1 IS (rx1_analyze,RX1_IDLE,RX1_ON,RX1_STALL);
        SIGNAL sreg1, next_sreg1 : type_sreg1;
        SIGNAL next_BP_rx0_add0,next_BP_rx0_add1,next_BP_rx0_add2,next_BP_rx0_add3,
                next_BP_rx0_add4,next_BP_rx0_add5,next_BP_rx0_add6,next_BP_rx0_add7,
                next_BP_rx0_cs0,next_BP_rx0_cs1,next_BP_rx0_cs2,next_BP_rx1_add0,
                next_BP_rx1_add1,next_BP_rx1_add2,next_BP_rx1_add3,next_BP_rx1_add4,
                next_BP_rx1_add5,next_BP_rx1_add6,next_BP_rx1_add7,next_BP_rx1_cs0,
                next_BP_rx1_cs1,next_BP_rx1_cs2,next_dest_tx00,next_dest_tx0_10,
                next_dest_tx0_11,next_dest_tx0_12,next_dest_tx0_13,next_dest_tx0_14,
                next_dest_tx0_15,next_dest_tx0_16,next_dest_tx0_17,next_dest_tx01,
                next_dest_tx1_10,next_dest_tx1_11,next_dest_tx1_12,next_dest_tx1_13,
                next_dest_tx1_14,next_dest_tx1_15,next_dest_tx1_16,next_dest_tx1_17,
                next_dest_tx02,next_dest_tx03,next_dest_tx04,next_dest_tx05,next_dest_tx06,
                next_dest_tx07,next_dest_tx10,next_dest_tx11,next_dest_tx12,next_dest_tx13,
                next_dest_tx14,next_dest_tx15,next_dest_tx16,next_dest_tx17,next_rx0_flag,
                next_rx1_flag : std_logic;
        SIGNAL BP_rx0_add : std_logic_vector (7 DOWNTO 0);
        SIGNAL BP_rx0_cs : std_logic_vector (2 DOWNTO 0);
        SIGNAL BP_rx1_add : std_logic_vector (7 DOWNTO 0);
        SIGNAL BP_rx1_cs : std_logic_vector (2 DOWNTO 0);
        SIGNAL dest_tx0 : std_logic_vector (7 DOWNTO 0);
        SIGNAL dest_tx0_1 : std_logic_vector (7 DOWNTO 0);
        SIGNAL dest_tx1 : std_logic_vector (7 DOWNTO 0);
        SIGNAL dest_tx1_1 : std_logic_vector (7 DOWNTO 0);
        SIGNAL rx0_add : std_logic_vector (7 DOWNTO 0);
        SIGNAL rx0_cs : std_logic_vector (2 DOWNTO 0);
        SIGNAL rx1_add : std_logic_vector (7 DOWNTO 0);
        SIGNAL rx1_cs : std_logic_vector (2 DOWNTO 0);
        SIGNAL tx0_mask : std_logic_vector (7 DOWNTO 0);
        SIGNAL tx1_mask : std_logic_vector (7 DOWNTO 0);

BEGIN
        PROCESS (CLK, RESET, next_sreg, next_rx0_flag, next_BP_rx0_add7,
                next_BP_rx0_add6, next_BP_rx0_add5, next_BP_rx0_add4, next_BP_rx0_add3,
                next_BP_rx0_add2, next_BP_rx0_add1, next_BP_rx0_add0, next_BP_rx0_cs2,
                next_BP_rx0_cs1, next_BP_rx0_cs0, next_dest_tx07, next_dest_tx06,
                next_dest_tx05, next_dest_tx04, next_dest_tx03, next_dest_tx02,
                next_dest_tx01, next_dest_tx00, next_dest_tx17, next_dest_tx16,
                next_dest_tx15, next_dest_tx14, next_dest_tx13, next_dest_tx12,
                next_dest_tx11, next_dest_tx10)
        BEGIN
                IF ( RESET='1' ) THEN
                        sreg <= RX0_IDLE;
                        rx0_flag <= '0';
                        BP_rx0_cs2 <= '0';
```

```
        BP_rx0_cs1 <= '0';
        BP_rx0_cs0 <= '0';
        dest_tx07 <= '0';
        dest_tx06 <= '0';
        dest_tx05 <= '0';
        dest_tx04 <= '0';
        dest_tx03 <= '0';
        dest_tx02 <= '0';
        dest_tx01 <= '0';
        dest_tx00 <= '0';
        dest_tx17 <= '0';
        dest_tx16 <= '0';
        dest_tx15 <= '0';
        dest_tx14 <= '0';
        dest_tx13 <= '0';
        dest_tx12 <= '0';
        dest_tx11 <= '0';
        dest_tx10 <= '0';
        BP_rx0_add7 <= '0';
        BP_rx0_add6 <= '0';
        BP_rx0_add5 <= '0';
        BP_rx0_add4 <= '0';
        BP_rx0_add3 <= '0';
        BP_rx0_add2 <= '0';
        BP_rx0_add1 <= '0';
        BP_rx0_add0 <= '0';
ELSIF CLK='1' AND CLK'event THEN
        sreg <= next_sreg;
        rx0_flag <= next_rx0_flag;
        BP_rx0_add7 <= next_BP_rx0_add7;
        BP_rx0_add6 <= next_BP_rx0_add6;
        BP_rx0_add5 <= next_BP_rx0_add5;
        BP_rx0_add4 <= next_BP_rx0_add4;
        BP_rx0_add3 <= next_BP_rx0_add3;
        BP_rx0_add2 <= next_BP_rx0_add2;
        BP_rx0_add1 <= next_BP_rx0_add1;
        BP_rx0_add0 <= next_BP_rx0_add0;
        BP_rx0_cs2 <= next_BP_rx0_cs2;
        BP_rx0_cs1 <= next_BP_rx0_cs1;
        BP_rx0_cs0 <= next_BP_rx0_cs0;
        dest_tx07 <= next_dest_tx07;
        dest_tx06 <= next_dest_tx06;
        dest_tx05 <= next_dest_tx05;
        dest_tx04 <= next_dest_tx04;
        dest_tx03 <= next_dest_tx03;
        dest_tx02 <= next_dest_tx02;
```

```
                dest_tx01 <= next_dest_tx01;
                dest_tx00 <= next_dest_tx00;
                dest_tx17 <= next_dest_tx17;
                dest_tx16 <= next_dest_tx16;
                dest_tx15 <= next_dest_tx15;
                dest_tx14 <= next_dest_tx14;
                dest_tx13 <= next_dest_tx13;
                dest_tx12 <= next_dest_tx12;
                dest_tx11 <= next_dest_tx11;
                dest_tx10 <= next_dest_tx10;
        END IF;
END PROCESS;

PROCESS (CLK, RESET, next_sreg1, next_rx1_flag, next_BP_rx1_add7,
        next_BP_rx1_add6, next_BP_rx1_add5, next_BP_rx1_add4, next_BP_rx1_add3,
        next_BP_rx1_add2, next_BP_rx1_add1, next_BP_rx1_add0, next_BP_rx1_cs2,
        next_BP_rx1_cs1, next_BP_rx1_cs0, next_dest_tx0_17, next_dest_tx0_16,
        next_dest_tx0_15, next_dest_tx0_14, next_dest_tx0_13, next_dest_tx0_12,
        next_dest_tx0_11, next_dest_tx0_10, next_dest_tx1_17, next_dest_tx1_16,
        next_dest_tx1_15, next_dest_tx1_14, next_dest_tx1_13, next_dest_tx1_12,
        next_dest_tx1_11, next_dest_tx1_10)
BEGIN
        IF ( RESET='1' ) THEN
                sreg1 <= RX1_IDLE;
                rx1_flag <= '0';
                BP_rx1_cs2 <= '0';
                BP_rx1_cs1 <= '0';
                BP_rx1_cs0 <= '0';
                dest_tx0_17 <= '0';
                dest_tx0_16 <= '0';
                dest_tx0_15 <= '0';
                dest_tx0_14 <= '0';
                dest_tx0_13 <= '0';
                dest_tx0_12 <= '0';
                dest_tx0_11 <= '0';
                dest_tx0_10 <= '0';
                dest_tx1_17 <= '0';
                dest_tx1_16 <= '0';
                dest_tx1_15 <= '0';
                dest_tx1_14 <= '0';
                dest_tx1_13 <= '0';
                dest_tx1_12 <= '0';
                dest_tx1_11 <= '0';
                dest_tx1_10 <= '0';
                BP_rx1_add7 <= '0';
                BP_rx1_add6 <= '0';
```

```
                BP_rx1_add5 <= '0';
                BP_rx1_add4 <= '0';
                BP_rx1_add3 <= '0';
                BP_rx1_add2 <= '0';
                BP_rx1_add1 <= '0';
                BP_rx1_add0 <= '0';
        ELSIF CLK='1' AND CLK'event THEN
                sreg1 <= next_sreg1;
                rx1_flag <= next_rx1_flag;
                BP_rx1_add7 <= next_BP_rx1_add7;
                BP_rx1_add6 <= next_BP_rx1_add6;
                BP_rx1_add5 <= next_BP_rx1_add5;
                BP_rx1_add4 <= next_BP_rx1_add4;
                BP_rx1_add3 <= next_BP_rx1_add3;
                BP_rx1_add2 <= next_BP_rx1_add2;
                BP_rx1_add1 <= next_BP_rx1_add1;
                BP_rx1_add0 <= next_BP_rx1_add0;
                BP_rx1_cs2 <= next_BP_rx1_cs2;
                BP_rx1_cs1 <= next_BP_rx1_cs1;
                BP_rx1_cs0 <= next_BP_rx1_cs0;
                dest_tx0_17 <= next_dest_tx0_17;
                dest_tx0_16 <= next_dest_tx0_16;
                dest_tx0_15 <= next_dest_tx0_15;
                dest_tx0_14 <= next_dest_tx0_14;
                dest_tx0_13 <= next_dest_tx0_13;
                dest_tx0_12 <= next_dest_tx0_12;
                dest_tx0_11 <= next_dest_tx0_11;
                dest_tx0_10 <= next_dest_tx0_10;
                dest_tx1_17 <= next_dest_tx1_17;
                dest_tx1_16 <= next_dest_tx1_16;
                dest_tx1_15 <= next_dest_tx1_15;
                dest_tx1_14 <= next_dest_tx1_14;
                dest_tx1_13 <= next_dest_tx1_13;
                dest_tx1_12 <= next_dest_tx1_12;
                dest_tx1_11 <= next_dest_tx1_11;
                dest_tx1_10 <= next_dest_tx1_10;
        END IF;
END PROCESS;

PROCESS (sreg,sreg1,BP_rx0_add0,BP_rx0_add1,BP_rx0_add2,BP_rx0_add3,

BP_rx0_add4,BP_rx0_add5,BP_rx0_add6,BP_rx0_add7,BP_rx0_cs0,BP_rx0_cs1,

BP_rx0_cs2,BP_rx1_add0,BP_rx1_add1,BP_rx1_add2,BP_rx1_add3,BP_rx1_add4,
        BP_rx1_add5,BP_rx1_add6,BP_rx1_add7,BP_rx1_cs0,BP_rx1_cs1,BP_rx1_cs2,
        dest_tx00,dest_tx0_10,dest_tx0_11,dest_tx0_12,dest_tx0_13,dest_tx0_14,
```

```
        dest_tx0_15,dest_tx0_16,dest_tx0_17,dest_tx01,dest_tx1_10,dest_tx1_11,
        dest_tx1_12,dest_tx1_13,dest_tx1_14,dest_tx1_15,dest_tx1_16,dest_tx1_17,
        dest_tx02,dest_tx03,dest_tx04,dest_tx05,dest_tx06,dest_tx07,dest_tx10,
        dest_tx11,dest_tx12,dest_tx13,dest_tx14,dest_tx15,dest_tx16,dest_tx17,
        rx0_dat0,rx0_dat1,rx0_dat2,rx0_dat3,rx0_dat4,rx0_dat5,rx0_dat6,rx0_dat7,
        rx0_eof,rx0_flag,rx0_sof,rx0_src_rdy,rx1_dat0,rx1_dat1,rx1_dat2,rx1_dat3,
        rx1_dat4,rx1_dat5,rx1_dat6,rx1_dat7,rx1_eof,rx1_flag,rx1_sof,rx1_src_rdy,
        tx0_mask0,tx0_mask1,tx0_mask2,tx0_mask3,tx0_mask4,tx0_mask5,tx0_mask6,
        tx0_mask7,tx1_mask0,tx1_mask1,tx1_mask2,tx1_mask3,tx1_mask4,tx1_mask5,
        tx1_mask6,tx1_mask7,BP_rx0_add,BP_rx0_cs,BP_rx1_add,BP_rx1_cs,dest_tx0,
        dest_tx0_1,dest_tx1,dest_tx1_1,bus_0_busy,bus_1_busy,bus_2_busy)
    BEGIN
        next_BP_rx0_add0 <= BP_rx0_add0;next_BP_rx0_add1 <= BP_rx0_add1;
            next_BP_rx0_add2 <= BP_rx0_add2;next_BP_rx0_add3 <=
BP_rx0_add3;
            next_BP_rx0_add4 <= BP_rx0_add4;next_BP_rx0_add5 <=
BP_rx0_add5;
            next_BP_rx0_add6 <= BP_rx0_add6;next_BP_rx0_add7 <=
BP_rx0_add7;
            next_BP_rx0_cs0 <= BP_rx0_cs0;next_BP_rx0_cs1 <=
BP_rx0_cs1;next_BP_rx0_cs2
            <= BP_rx0_cs2;next_BP_rx1_add0 <= BP_rx1_add0;next_BP_rx1_add1
<=
            BP_rx1_add1;next_BP_rx1_add2 <= BP_rx1_add2;next_BP_rx1_add3
<= BP_rx1_add3;
            next_BP_rx1_add4 <= BP_rx1_add4;next_BP_rx1_add5 <=
BP_rx1_add5;
            next_BP_rx1_add6 <= BP_rx1_add6;next_BP_rx1_add7 <=
BP_rx1_add7;
            next_BP_rx1_cs0 <= BP_rx1_cs0;next_BP_rx1_cs1 <=
BP_rx1_cs1;next_BP_rx1_cs2
            <= BP_rx1_cs2;next_dest_tx00 <= dest_tx00;next_dest_tx0_10 <=
dest_tx0_10;
            next_dest_tx0_11 <= dest_tx0_11;next_dest_tx0_12 <= dest_tx0_12;
            next_dest_tx0_13 <= dest_tx0_13;next_dest_tx0_14 <= dest_tx0_14;
            next_dest_tx0_15 <= dest_tx0_15;next_dest_tx0_16 <= dest_tx0_16;
            next_dest_tx0_17 <= dest_tx0_17;next_dest_tx01 <=
dest_tx01;next_dest_tx1_10
            <= dest_tx1_10;next_dest_tx1_11 <= dest_tx1_11;next_dest_tx1_12 <=
            dest_tx1_12;next_dest_tx1_13 <= dest_tx1_13;next_dest_tx1_14 <=
dest_tx1_14;
            next_dest_tx1_15 <= dest_tx1_15;next_dest_tx1_16 <= dest_tx1_16;
            next_dest_tx1_17 <= dest_tx1_17;next_dest_tx02 <=
dest_tx02;next_dest_tx03 <=
             dest_tx03;next_dest_tx04 <= dest_tx04;next_dest_tx05 <= dest_tx05;
```

```
                    next_dest_tx06 <= dest_tx06;next_dest_tx07 <= dest_tx07;next_dest_tx10
<=
                    dest_tx10;next_dest_tx11 <= dest_tx11;next_dest_tx12 <= dest_tx12;
                    next_dest_tx13 <= dest_tx13;next_dest_tx14 <= dest_tx14;next_dest_tx15
<=
                    dest_tx15;next_dest_tx16 <= dest_tx16;next_dest_tx17 <= dest_tx17;
                    next_rx0_flag <= rx0_flag;next_rx1_flag <= rx1_flag;

                BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7, BP_rx0_add6, BP_rx0_add5,
                        BP_rx0_add4, BP_rx0_add3, BP_rx0_add2, BP_rx0_add1,
BP_rx0_add0)));
                BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2, BP_rx0_cs1, BP_rx0_cs0)));
                BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7, BP_rx1_add6, BP_rx1_add5,
                        BP_rx1_add4, BP_rx1_add3, BP_rx1_add2, BP_rx1_add1,
BP_rx1_add0)));
                BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2, BP_rx1_cs1, BP_rx1_cs0)));
                dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06, dest_tx05, dest_tx04
                        , dest_tx03, dest_tx02, dest_tx01, dest_tx00)));
                dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17, dest_tx0_16, dest_tx0_15,
                        dest_tx0_14, dest_tx0_13, dest_tx0_12, dest_tx0_11, dest_tx0_10)));
                dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16, dest_tx15, dest_tx14
                        , dest_tx13, dest_tx12, dest_tx11, dest_tx10)));
                dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17, dest_tx1_16, dest_tx1_15,
                        dest_tx1_14, dest_tx1_13, dest_tx1_12, dest_tx1_11, dest_tx1_10)));


                next_sreg<=RX0_IDLE;
                next_sreg1<=RX1_IDLE;

                CASE sreg IS
                        WHEN rx0_analyze =>
                                next_sreg<=RX0_ON;
                                next_rx0_flag<='1';

                                dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06, dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01, dest_tx00)));
                                dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16, dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11, dest_tx10)));
                                --My modifications
                                IF (bus_0_busy='0') THEN
BP_rx0_cs<=(std_logic_vector'("001"));
                                ELSIF (bus_1_busy='0') THEN
BP_rx0_cs<=(std_logic_vector'("010"));
                                ELSIF (bus_2_busy='0') THEN
BP_rx0_cs<=(std_logic_vector'("100"));
                                ELSE BP_rx0_cs<=(std_logic_vector'("000"));
```

```
                        END IF;

                        IF ((dest_tx00 = '1') OR (dest_tx01 = '1') OR (dest_tx02 = '1') OR
(dest_tx03 = '1')
                                OR (dest_tx04 = '1') OR (dest_tx05 = '1') OR (dest_tx06 =
'1') OR (dest_tx07 = '1')) THEN
                                BP_rx0_add<=(( std_logic_vector("00000000")));
                        ELSIF ((dest_tx10 = '1') OR (dest_tx11 = '1') OR (dest_tx12 = '1')
OR (dest_tx13 = '1')
                                OR (dest_tx14 = '1') OR (dest_tx15 = '1') OR (dest_tx16 =
'1') OR (dest_tx17 = '1')) THEN
                                BP_rx0_add<=(( std_logic_vector("00100000")));
                        ELSE BP_rx0_add<=(( std_logic_vector("01000000")));
                        END IF;
                        ------------------
                WHEN RX0_IDLE =>
                        IF ( rx0_sof='0' ) THEN
                                next_sreg<=rx0_analyze;
                                BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));

                                BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                        , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)));
                                IF (( rx0_flag='1' )) THEN next_rx0_flag<='1';
                                ELSE next_rx0_flag<='0';
                                END IF;

                                dest_tx1 <= (( std_logic_vector'(rx0_dat7, rx0_dat6,
rx0_dat5, rx0_dat4,
                                         rx0_dat3, rx0_dat2, rx0_dat1, rx0_dat0)) AND (
std_logic_vector'(tx1_mask7,
                                        tx1_mask6, tx1_mask5, tx1_mask4, tx1_mask3,
tx1_mask2, tx1_mask1, tx1_mask0))
                                        );
                                dest_tx0 <= (( std_logic_vector'(rx0_dat7, rx0_dat6,
rx0_dat5, rx0_dat4,
                                         rx0_dat3, rx0_dat2, rx0_dat1, rx0_dat0)) AND (
std_logic_vector'(tx0_mask7,
                                        tx0_mask6, tx0_mask5, tx0_mask4, tx0_mask3,
tx0_mask2, tx0_mask1, tx0_mask0))
                                        );
                         ELSE
                                next_sreg<=RX0_IDLE;
                                next_rx0_flag<='0';
```

```
                                    dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                    dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                    BP_rx0_cs <= (( std_logic_vector'("000")));
                                    BP_rx0_add <= (std_logic_vector'("00000000"));
                                END IF;
                            WHEN RX0_ON =>
                                IF ( rx0_src_rdy='0' AND rx0_eof='1' ) OR ( rx0_eof='0' AND
                                    rx0_src_rdy='1' ) THEN
                                    next_sreg<=RX0_ON;
                                    next_rx0_flag<='1';

                                    dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                    dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                    BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));
                                    BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                        , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)) +
                                        std_logic_vector'("00000001"));
                                END IF;
                                IF ( rx0_src_rdy='1' AND rx0_eof='1') THEN
                                    next_sreg<=RX0_STALL;

                                    dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                    dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
```

```
                                BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));
                                IF (( rx0_flag='1' )) THEN next_rx0_flag<='1';
                                ELSE next_rx0_flag<='0';
                                END IF;

                                BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                        , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)));
                        END IF;
                        IF ( rx0_eof='0' AND rx0_src_rdy='0') THEN
                                next_sreg<=RX0_IDLE;
                                next_rx0_flag<='0';

                                dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));
                                BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                        , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)) +
                                        std_logic_vector'("00000001"));
                        END IF;
                WHEN RX0_STALL =>
                        IF ( rx0_src_rdy='0' ) THEN
                                next_sreg<=RX0_ON;
                                next_rx0_flag<='1';

                                dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));
```

```
                                        BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                                , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)) +
                                                std_logic_vector'("00000001"));
                                ELSE
                                        next_sreg<=RX0_STALL;

                                        dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                                dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                        dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                                dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                        BP_rx0_cs <= (( std_logic_vector'(BP_rx0_cs2,
BP_rx0_cs1, BP_rx0_cs0)));
                                        IF (( rx0_flag='1' )) THEN next_rx0_flag<='1';
                                        ELSE next_rx0_flag<='0';
                                        END IF;

                                        BP_rx0_add <= (( std_logic_vector'(BP_rx0_add7,
BP_rx0_add6, BP_rx0_add5
                                                , BP_rx0_add4, BP_rx0_add3, BP_rx0_add2,
BP_rx0_add1, BP_rx0_add0)));
                                END IF;
                        WHEN OTHERS =>
                END CASE;

                CASE sreg1 IS
                        WHEN rx1_analyze =>
                                next_sreg1<=RX1_ON;
                                next_rx1_flag<='1';

                                dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17, dest_tx0_16,
dest_tx0_15,
                                         dest_tx0_14, dest_tx0_13, dest_tx0_12, dest_tx0_11,
dest_tx0_10)));
                                dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17, dest_tx1_16,
dest_tx1_15,
                                         dest_tx1_14, dest_tx1_13, dest_tx1_12, dest_tx1_11,
dest_tx1_10)));
                                IF (bus_0_busy='0') THEN
BP_rx1_cs<=(std_logic_vector'("001"));
```

```
                        ELSIF (bus_1_busy='0') THEN
BP_rx1_cs<=(std_logic_vector("010"));
                        ELSIF (bus_2_busy='0') THEN
BP_rx1_cs<=(std_logic_vector("100"));
                        ELSE BP_rx1_cs<=(std_logic_vector("000"));
                        END IF;
                        --My modifications
                        IF ((dest_tx0_10 = '1') OR (dest_tx0_11 = '1') OR (dest_tx0_12 =
'1') OR (dest_tx0_13 = '1')
                                OR (dest_tx0_14 = '1') OR (dest_tx0_15 = '1') OR
(dest_tx0_16 = '1') OR (dest_tx0_17 = '1')) THEN
                                BP_rx1_add<=(( std_logic_vector("00000000")));
                        ELSIF ((dest_tx1_10 = '1') OR (dest_tx1_11 = '1') OR
(dest_tx1_12 = '1') OR (dest_tx1_13 = '1')
                                OR (dest_tx1_14 = '1') OR (dest_tx1_15 = '1') OR
(dest_tx1_16 = '1') OR (dest_tx1_17 = '1')) THEN
                                BP_rx1_add<=(( std_logic_vector("00100000")));
                        ELSE BP_rx1_add<=(( std_logic_vector("01000000")));
                        END IF;
                        ------------------
                WHEN RX1_IDLE =>
                        IF ( rx1_sof='0' ) THEN
                                next_sreg1<=rx1_analyze;
                                BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));

                                BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                        , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)));
                                IF (( rx1_flag='1' )) THEN next_rx1_flag<='1';
                                ELSE next_rx1_flag<='0';
                                END IF;

                                dest_tx1_1 <= (( std_logic_vector'(rx1_dat7, rx1_dat6,
rx1_dat5,
                                        rx1_dat4, rx1_dat3, rx1_dat2, rx1_dat1, rx1_dat0))
AND ( std_logic_vector'(
                                        tx1_mask7, tx1_mask6, tx1_mask5, tx1_mask4,
tx1_mask3, tx1_mask2, tx1_mask1,
                                        tx1_mask0)));
                                dest_tx0_1 <= (( std_logic_vector'(rx1_dat7, rx1_dat6,
rx1_dat5,
                                        rx1_dat4, rx1_dat3, rx1_dat2, rx1_dat1, rx1_dat0))
AND ( std_logic_vector'(
```

```
                                        tx0_mask7, tx0_mask6, tx0_mask5, tx0_mask4,
tx0_mask3, tx0_mask2, tx0_mask1,
                                        tx0_mask0)));
                                ELSE
                                        next_sreg1<=RX1_IDLE;
                                        next_rx1_flag<='0';

                                        dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
                                                , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
                                        dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                                , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                        BP_rx1_cs <= (( std_logic_vector'("000")));
                                        BP_rx1_add <= (std_logic_vector'("00000000"));
                                END IF;
                        WHEN RX1_ON =>
                                IF ( rx1_src_rdy='0' AND rx1_eof='1' ) OR ( rx1_eof='0' AND
                                        rx1_src_rdy='1' ) THEN
                                        next_sreg1<=RX1_ON;
                                        next_rx1_flag<='1';

                                        dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
                                                , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
                                        dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                                , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                        BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));
                                        BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                                , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)) +
                                                std_logic_vector'("00000001"));
                                END IF;
                                IF ( rx1_src_rdy='1' AND rx1_eof='1' ) THEN
                                        next_sreg1<=RX1_STALL;

                                        dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
```

```
                                        , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
                                dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                        , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));
                                IF (( rx1_flag='1' )) THEN next_rx1_flag<='1';
                                ELSE next_rx1_flag<='0';
                                END IF;

                                BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                        , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)));
                        END IF;
                        IF ( rx1_eof='0' AND rx1_src_rdy='0') THEN
                                next_sreg1<=RX1_IDLE;
                                next_rx1_flag<='0';

                                dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
                                        , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
                                dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                        , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));
                                BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                        , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)) +
                                        std_logic_vector'("00000001"));
                        END IF;
                WHEN RX1_STALL =>
                        IF ( rx1_src_rdy='0' ) THEN
                                next_sreg1<=RX1_ON;
                                next_rx1_flag<='1';

                                dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
                                        , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
```

```
                                        dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                                , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                        BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));
                                        BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                                , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)) +
                                                std_logic_vector'("00000001"));
                                ELSE
                                        next_sreg1<=RX1_STALL;

                                        dest_tx0_1 <= (( std_logic_vector'(dest_tx0_17,
dest_tx0_16, dest_tx0_15
                                                , dest_tx0_14, dest_tx0_13, dest_tx0_12,
dest_tx0_11, dest_tx0_10)));
                                        dest_tx1_1 <= (( std_logic_vector'(dest_tx1_17,
dest_tx1_16, dest_tx1_15
                                                , dest_tx1_14, dest_tx1_13, dest_tx1_12,
dest_tx1_11, dest_tx1_10)));
                                        BP_rx1_cs <= (( std_logic_vector'(BP_rx1_cs2,
BP_rx1_cs1, BP_rx1_cs0)));
                                        IF (( rx1_flag='1' )) THEN next_rx1_flag<='1';
                                        ELSE next_rx1_flag<='0';
                                        END IF;

                                        BP_rx1_add <= (( std_logic_vector'(BP_rx1_add7,
BP_rx1_add6, BP_rx1_add5
                                                , BP_rx1_add4, BP_rx1_add3, BP_rx1_add2,
BP_rx1_add1, BP_rx1_add0)));
                                END IF;
                        WHEN OTHERS =>
                END CASE;

                next_BP_rx0_add7 <= BP_rx0_add(7);
                next_BP_rx0_add6 <= BP_rx0_add(6);
                next_BP_rx0_add5 <= BP_rx0_add(5);
                next_BP_rx0_add4 <= BP_rx0_add(4);
                next_BP_rx0_add3 <= BP_rx0_add(3);
                next_BP_rx0_add2 <= BP_rx0_add(2);
                next_BP_rx0_add1 <= BP_rx0_add(1);
                next_BP_rx0_add0 <= BP_rx0_add(0);
                next_BP_rx0_cs2 <= BP_rx0_cs(2);
                next_BP_rx0_cs1 <= BP_rx0_cs(1);
```

```
        next_BP_rx0_cs0 <= BP_rx0_cs(0);
        next_BP_rx1_add7 <= BP_rx1_add(7);
        next_BP_rx1_add6 <= BP_rx1_add(6);
        next_BP_rx1_add5 <= BP_rx1_add(5);
        next_BP_rx1_add4 <= BP_rx1_add(4);
        next_BP_rx1_add3 <= BP_rx1_add(3);
        next_BP_rx1_add2 <= BP_rx1_add(2);
        next_BP_rx1_add1 <= BP_rx1_add(1);
        next_BP_rx1_add0 <= BP_rx1_add(0);
        next_BP_rx1_cs2 <= BP_rx1_cs(2);
        next_BP_rx1_cs1 <= BP_rx1_cs(1);
        next_BP_rx1_cs0 <= BP_rx1_cs(0);
        next_dest_tx07 <= dest_tx0(7);
        next_dest_tx06 <= dest_tx0(6);
        next_dest_tx05 <= dest_tx0(5);
        next_dest_tx04 <= dest_tx0(4);
        next_dest_tx03 <= dest_tx0(3);
        next_dest_tx02 <= dest_tx0(2);
        next_dest_tx01 <= dest_tx0(1);
        next_dest_tx00 <= dest_tx0(0);
        next_dest_tx0_17 <= dest_tx0_1(7);
        next_dest_tx0_16 <= dest_tx0_1(6);
        next_dest_tx0_15 <= dest_tx0_1(5);
        next_dest_tx0_14 <= dest_tx0_1(4);
        next_dest_tx0_13 <= dest_tx0_1(3);
        next_dest_tx0_12 <= dest_tx0_1(2);
        next_dest_tx0_11 <= dest_tx0_1(1);
        next_dest_tx0_10 <= dest_tx0_1(0);
        next_dest_tx17 <= dest_tx1(7);
        next_dest_tx16 <= dest_tx1(6);
        next_dest_tx15 <= dest_tx1(5);
        next_dest_tx14 <= dest_tx1(4);
        next_dest_tx13 <= dest_tx1(3);
        next_dest_tx12 <= dest_tx1(2);
        next_dest_tx11 <= dest_tx1(1);
        next_dest_tx10 <= dest_tx1(0);
        next_dest_tx1_17 <= dest_tx1_1(7);
        next_dest_tx1_16 <= dest_tx1_1(6);
        next_dest_tx1_15 <= dest_tx1_1(5);
        next_dest_tx1_14 <= dest_tx1_1(4);
        next_dest_tx1_13 <= dest_tx1_1(3);
        next_dest_tx1_12 <= dest_tx1_1(2);
        next_dest_tx1_11 <= dest_tx1_1(1);
        next_dest_tx1_10 <= dest_tx1_1(0);
END PROCESS;
```

```
	PROCESS (rx0_flag,rx1_flag)
	BEGIN
		IF (( rx0_flag='1' ) OR ( rx1_flag='1' )) THEN rx_flag<='1';
		ELSE rx_flag<='0';
		END IF;
	END PROCESS;

	PROCESS (BP_rx0_add0,BP_rx0_add1,BP_rx0_add2,BP_rx0_add3,BP_rx0_add4,
		BP_rx0_add5,BP_rx0_add6,BP_rx0_add7,rx0_add)
	BEGIN
		rx0_add <= (( std_logic_vector'(BP_rx0_add7, BP_rx0_add6, BP_rx0_add5,
			BP_rx0_add4, BP_rx0_add3, BP_rx0_add2, BP_rx0_add1,
BP_rx0_add0)));
		rx0_add0 <= rx0_add(0);
		rx0_add1 <= rx0_add(1);
		rx0_add2 <= rx0_add(2);
		rx0_add3 <= rx0_add(3);
		rx0_add4 <= rx0_add(4);
		rx0_add5 <= rx0_add(5);
		rx0_add6 <= rx0_add(6);
		rx0_add7 <= rx0_add(7);
	END PROCESS;

	PROCESS (BP_rx0_cs0,BP_rx0_cs1,BP_rx0_cs2,rx0_cs)
	BEGIN
		rx0_cs <= (( std_logic_vector'(BP_rx0_cs2, BP_rx0_cs1, BP_rx0_cs0)));
		rx0_cs0 <= rx0_cs(0);
		rx0_cs1 <= rx0_cs(1);
		rx0_cs2 <= rx0_cs(2);
	END PROCESS;

	PROCESS (BP_rx1_add0,BP_rx1_add1,BP_rx1_add2,BP_rx1_add3,BP_rx1_add4,
		BP_rx1_add5,BP_rx1_add6,BP_rx1_add7,rx1_add)
	BEGIN
		rx1_add <= (( std_logic_vector'(BP_rx1_add7, BP_rx1_add6, BP_rx1_add5,
			BP_rx1_add4, BP_rx1_add3, BP_rx1_add2, BP_rx1_add1,
BP_rx1_add0)));
		rx1_add0 <= rx1_add(0);
		rx1_add1 <= rx1_add(1);
		rx1_add2 <= rx1_add(2);
		rx1_add3 <= rx1_add(3);
		rx1_add4 <= rx1_add(4);
		rx1_add5 <= rx1_add(5);
		rx1_add6 <= rx1_add(6);
		rx1_add7 <= rx1_add(7);
	END PROCESS;
```

```
	PROCESS (BP_rx1_cs0,BP_rx1_cs1,BP_rx1_cs2,rx1_cs)
	BEGIN
		rx1_cs <= (( std_logic_vector'(BP_rx1_cs2, BP_rx1_cs1, BP_rx1_cs0)));
		rx1_cs0 <= rx1_cs(0);
		rx1_cs1 <= rx1_cs(1);
		rx1_cs2 <= rx1_cs(2);
	END PROCESS;

	PROCESS (tx0_mask)
	BEGIN
		tx0_mask <= (std_logic_vector'("00001110"));
		tx0_mask0 <= tx0_mask(0);
		tx0_mask1 <= tx0_mask(1);
		tx0_mask2 <= tx0_mask(2);
		tx0_mask3 <= tx0_mask(3);
		tx0_mask4 <= tx0_mask(4);
		tx0_mask5 <= tx0_mask(5);
		tx0_mask6 <= tx0_mask(6);
		tx0_mask7 <= tx0_mask(7);
	END PROCESS;

	PROCESS (tx1_mask)
	BEGIN
		tx1_mask <= (std_logic_vector'("11110000"));
		tx1_mask0 <= tx1_mask(0);
		tx1_mask1 <= tx1_mask(1);
		tx1_mask2 <= tx1_mask(2);
		tx1_mask3 <= tx1_mask(3);
		tx1_mask4 <= tx1_mask(4);
		tx1_mask5 <= tx1_mask(5);
		tx1_mask6 <= tx1_mask(6);
		tx1_mask7 <= tx1_mask(7);
	END PROCESS;
END BEHAVIOR;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY RECV IS
	PORT (rx0_add : OUT std_logic_vector (7 DOWNTO 0);
		rx0_cs : OUT std_logic_vector (2 DOWNTO 0);
		rx0_dat : IN std_logic_vector (7 DOWNTO 0);
```

```
                rx1_add : OUT std_logic_vector (7 DOWNTO 0);
                rx1_cs : OUT std_logic_vector (2 DOWNTO 0);
                rx1_dat : IN std_logic_vector (7 DOWNTO 0);
                CLK,RESET,rx0_eof,rx0_sof,rx0_src_rdy,rx1_eof,rx1_sof,rx1_src_rdy,
                bus_0_busy,bus_1_busy,bus_2_busy: IN std_logic;
                rx_flag : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF RECV IS
        COMPONENT SHELL_RECV
                PORT (CLK,RESET,rx0_dat0,rx0_dat1,rx0_dat2,rx0_dat3,rx0_dat4,rx0_dat5,

        rx0_dat6,rx0_dat7,rx0_eof,rx0_sof,rx0_src_rdy,rx1_dat0,rx1_dat1,rx1_dat2,

        rx1_dat3,rx1_dat4,rx1_dat5,rx1_dat6,rx1_dat7,rx1_eof,rx1_sof,rx1_src_rdy,
                        bus_0_busy,bus_1_busy,bus_2_busy: IN std_logic;

        rx0_add0,rx0_add1,rx0_add2,rx0_add3,rx0_add4,rx0_add5,rx0_add6,rx0_add7,

        rx0_cs0,rx0_cs1,rx0_cs2,rx1_add0,rx1_add1,rx1_add2,rx1_add3,rx1_add4,rx1_add5
                        ,rx1_add6,rx1_add7,rx1_cs0,rx1_cs1,rx1_cs2,rx_flag : OUT std_logic);
        END COMPONENT;
BEGIN
        SHELL1_RECV : SHELL_RECV PORT MAP
(CLK=>CLK,RESET=>RESET,rx0_dat0=>rx0_dat(0
                ),rx0_dat1=>rx0_dat(1),rx0_dat2=>rx0_dat(2),rx0_dat3=>rx0_dat(3),rx0_dat4=>
                rx0_dat(4),rx0_dat5=>rx0_dat(5),rx0_dat6=>rx0_dat(6),rx0_dat7=>rx0_dat(7),

        rx0_eof=>rx0_eof,rx0_sof=>rx0_sof,rx0_src_rdy=>rx0_src_rdy,rx1_dat0=>rx1_dat(

        0),rx1_dat1=>rx1_dat(1),rx1_dat2=>rx1_dat(2),rx1_dat3=>rx1_dat(3),rx1_dat4=>
                rx1_dat(4),rx1_dat5=>rx1_dat(5),rx1_dat6=>rx1_dat(6),rx1_dat7=>rx1_dat(7),

        rx1_eof=>rx1_eof,rx1_sof=>rx1_sof,rx1_src_rdy=>rx1_src_rdy,rx0_add0=>rx0_add(

        0),rx0_add1=>rx0_add(1),rx0_add2=>rx0_add(2),rx0_add3=>rx0_add(3),rx0_add4=>

        rx0_add(4),rx0_add5=>rx0_add(5),rx0_add6=>rx0_add(6),rx0_add7=>rx0_add(7),

        rx0_cs0=>rx0_cs(0),rx0_cs1=>rx0_cs(1),rx0_cs2=>rx0_cs(2),rx1_add0=>rx1_add(0)

        ,rx1_add1=>rx1_add(1),rx1_add2=>rx1_add(2),rx1_add3=>rx1_add(3),rx1_add4=>

        rx1_add(4),rx1_add5=>rx1_add(5),rx1_add6=>rx1_add(6),rx1_add7=>rx1_add(7),
                rx1_cs0=>rx1_cs(0),rx1_cs1=>rx1_cs(1),rx1_cs2=>rx1_cs(2),rx_flag=>rx_flag,
```

```
        bus_0_busy=>bus_0_busy,bus_1_busy=>bus_1_busy,bus_2_busy=>bus_2_busy);
END BEHAVIOR;
```

Transmit State Machine:

```
-- C:\XILINX\TRIDENT\TXTEST\TX2SMTEST\TRANS.vhd
-- VHDL code created by Xilinx's StateCAD 6.2i
-- Tue Apr 12 23:26:17 2005

-- This VHDL code (for use with IEEE compliant tools) was generated using:
-- enumerated state assignment with structured code format.
-- Minimization is enabled,  implied else is enabled,
-- and outputs are speed optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY SHELL_TRANS IS
        PORT (CLK,bus_status0,bus_status1,bus_status2,RESET,tx0_add_en,tx0_dat0,
                tx0_dat1,tx0_dat2,tx0_dat3,tx0_dat4,tx0_dat5,tx0_dat6,tx0_dat7,tx0_new_bus0,
                tx0_new_bus1,tx0_new_bus2,tx1_add_en,tx1_dat0,tx1_dat1,tx1_dat2,tx1_dat3,

        tx1_dat4,tx1_dat5,tx1_dat6,tx1_dat7,tx1_new_bus0,tx1_new_bus1,tx1_new_bus2:
                IN std_logic;
                no_tx0,no_tx1,tx0_add0,tx0_add1,tx0_add2,tx0_add3,tx0_add4,tx0_add5,

        tx0_add6,tx0_add7,tx0_bus0,tx0_bus1,tx0_bus2,tx0_eof,tx0_sof,tx0_src_rdy,

        tx1_add0,tx1_add1,tx1_add2,tx1_add3,tx1_add4,tx1_add5,tx1_add6,tx1_add7,
                        tx1_bus0,tx1_bus1,tx1_bus2,tx1_eof,tx1_sof,tx1_src_rdy : OUT
std_logic);

        SIGNAL
BP_no_tx0,BP_no_tx1,BP_tx0_add0,BP_tx0_add1,BP_tx0_add2,BP_tx0_add3,

        BP_tx0_add4,BP_tx0_add5,BP_tx0_add6,BP_tx0_add7,BP_tx0_bus0,BP_tx0_bus1,

        BP_tx0_bus2,BP_tx0_eof,BP_tx0_sof,BP_tx0_src_rdy,BP_tx1_add0,BP_tx1_add1,

        BP_tx1_add2,BP_tx1_add3,BP_tx1_add4,BP_tx1_add5,BP_tx1_add6,BP_tx1_add7,

        BP_tx1_bus0,BP_tx1_bus1,BP_tx1_bus2,BP_tx1_eof,BP_tx1_sof,BP_tx1_src_rdy,
                bus0_na0,bus0_na1,bus0_na2,bus_na0,bus_na1,bus_na2,tx0_next_bus0,
                tx0_next_bus1,tx0_next_bus2,tx1_next_bus0,tx1_next_bus1,tx1_next_bus2:
                std_logic;
END;
```

```
ARCHITECTURE BEHAVIOR OF SHELL_TRANS IS
	TYPE type_sreg IS (START_TX0,TX0_IDLE,TX0_ON,END_TX0,TX0_SETUP);
	SIGNAL sreg, next_sreg : type_sreg;
	TYPE type_sreg1 IS (START_TX1,TX1_IDLE,TX1_ON,END_TX1,TX1_SETUP);
	SIGNAL sreg1, next_sreg1 : type_sreg1;
	SIGNAL next_BP_no_tx0,next_BP_no_tx1,next_BP_tx0_add0,next_BP_tx0_add1,
		next_BP_tx0_add2,next_BP_tx0_add3,next_BP_tx0_add4,next_BP_tx0_add5,
		next_BP_tx0_add6,next_BP_tx0_add7,next_BP_tx0_bus0,next_BP_tx0_bus1,
		next_BP_tx0_bus2,next_BP_tx0_eof,next_BP_tx0_sof,next_BP_tx0_src_rdy,
		next_BP_tx1_add0,next_BP_tx1_add1,next_BP_tx1_add2,next_BP_tx1_add3,
		next_BP_tx1_add4,next_BP_tx1_add5,next_BP_tx1_add6,next_BP_tx1_add7,
		next_BP_tx1_bus0,next_BP_tx1_bus1,next_BP_tx1_bus2,next_BP_tx1_eof,

	next_BP_tx1_sof,next_BP_tx1_src_rdy,next_bus0_na0,next_bus0_na1,next_bus0_na2

	,next_bus_na0,next_bus_na1,next_bus_na2,next_tx0_next_bus0,next_tx0_next_bus1

	,next_tx0_next_bus2,next_tx1_next_bus0,next_tx1_next_bus1,next_tx1_next_bus2
		: std_logic;
	SIGNAL BP_tx0_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL BP_tx0_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL BP_tx1_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL BP_tx1_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL bus0_na : std_logic_vector (2 DOWNTO 0);
	SIGNAL bus_na : std_logic_vector (2 DOWNTO 0);
	SIGNAL tx0_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL tx0_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL tx0_next_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL tx1_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL tx1_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL tx1_next_bus : std_logic_vector (2 DOWNTO 0);
BEGIN
	PROCESS (CLK, RESET, next_sreg, next_BP_no_tx0, next_BP_tx0_eof,
		next_BP_tx0_sof, next_BP_tx0_src_rdy, next_BP_tx0_add7, next_BP_tx0_add6,
		next_BP_tx0_add5, next_BP_tx0_add4, next_BP_tx0_add3, next_BP_tx0_add2,
		next_BP_tx0_add1, next_BP_tx0_add0, next_BP_tx0_bus2, next_BP_tx0_bus1,
		next_BP_tx0_bus0, next_bus0_na2, next_bus0_na1, next_bus0_na0,
		next_tx0_next_bus2, next_tx0_next_bus1, next_tx0_next_bus0)
	BEGIN
		IF ( RESET='1' ) THEN
			sreg <= TX0_IDLE;
			bus0_na2 <= '0';
			bus0_na1 <= '0';
			bus0_na0 <= '0';
			tx0_next_bus2 <= '0';
```

```
                tx0_next_bus1 <= '0';
                tx0_next_bus0 <= '0';
                BP_no_tx0 <= '1';
                BP_tx0_eof <= '1';
                BP_tx0_sof <= '1';
                BP_tx0_src_rdy <= '1';
                BP_tx0_add7 <= '0';
                BP_tx0_add6 <= '0';
                BP_tx0_add5 <= '0';
                BP_tx0_add4 <= '0';
                BP_tx0_add3 <= '0';
                BP_tx0_add2 <= '0';
                BP_tx0_add1 <= '0';
                BP_tx0_add0 <= '0';
                BP_tx0_bus2 <= '0';
                BP_tx0_bus1 <= '0';
                BP_tx0_bus0 <= '0';
        ELSIF CLK='1' AND CLK'event THEN
                sreg <= next_sreg;
                BP_no_tx0 <= next_BP_no_tx0;
                BP_tx0_eof <= next_BP_tx0_eof;
                BP_tx0_sof <= next_BP_tx0_sof;
                BP_tx0_src_rdy <= next_BP_tx0_src_rdy;
                BP_tx0_add7 <= next_BP_tx0_add7;
                BP_tx0_add6 <= next_BP_tx0_add6;
                BP_tx0_add5 <= next_BP_tx0_add5;
                BP_tx0_add4 <= next_BP_tx0_add4;
                BP_tx0_add3 <= next_BP_tx0_add3;
                BP_tx0_add2 <= next_BP_tx0_add2;
                BP_tx0_add1 <= next_BP_tx0_add1;
                BP_tx0_add0 <= next_BP_tx0_add0;
                BP_tx0_bus2 <= next_BP_tx0_bus2;
                BP_tx0_bus1 <= next_BP_tx0_bus1;
                BP_tx0_bus0 <= next_BP_tx0_bus0;
                bus0_na2 <= next_bus0_na2;
                bus0_na1 <= next_bus0_na1;
                bus0_na0 <= next_bus0_na0;
                tx0_next_bus2 <= next_tx0_next_bus2;
                tx0_next_bus1 <= next_tx0_next_bus1;
                tx0_next_bus0 <= next_tx0_next_bus0;
        END IF;
END PROCESS;

PROCESS (CLK, RESET, next_sreg1, next_BP_no_tx1, next_BP_tx1_eof,
        next_BP_tx1_sof, next_BP_tx1_src_rdy, next_BP_tx1_add7, next_BP_tx1_add6,
        next_BP_tx1_add5, next_BP_tx1_add4, next_BP_tx1_add3, next_BP_tx1_add2,
```

```
        next_BP_tx1_add1, next_BP_tx1_add0, next_BP_tx1_bus2, next_BP_tx1_bus1,
        next_BP_tx1_bus0, next_bus_na2, next_bus_na1, next_bus_na0,
        next_tx1_next_bus2, next_tx1_next_bus1, next_tx1_next_bus0)
BEGIN
        IF ( RESET='1' ) THEN
                sreg1 <= TX1_IDLE;
                bus_na2 <= '0';
                bus_na1 <= '0';
                bus_na0 <= '0';
                tx1_next_bus2 <= '0';
                tx1_next_bus1 <= '0';
                tx1_next_bus0 <= '0';
                BP_no_tx1 <= '1';
                BP_tx1_eof <= '1';
                BP_tx1_sof <= '1';
                BP_tx1_src_rdy <= '1';
                BP_tx1_add7 <= '0';
                BP_tx1_add6 <= '0';
                BP_tx1_add5 <= '1';
                BP_tx1_add4 <= '0';
                BP_tx1_add3 <= '0';
                BP_tx1_add2 <= '0';
                BP_tx1_add1 <= '0';
                BP_tx1_add0 <= '0';
                BP_tx1_bus2 <= '0';
                BP_tx1_bus1 <= '0';
                BP_tx1_bus0 <= '0';
        ELSIF CLK='1' AND CLK'event THEN
                sreg1 <= next_sreg1;
                BP_no_tx1 <= next_BP_no_tx1;
                BP_tx1_eof <= next_BP_tx1_eof;
                BP_tx1_sof <= next_BP_tx1_sof;
                BP_tx1_src_rdy <= next_BP_tx1_src_rdy;
                BP_tx1_add7 <= next_BP_tx1_add7;
                BP_tx1_add6 <= next_BP_tx1_add6;
                BP_tx1_add5 <= next_BP_tx1_add5;
                BP_tx1_add4 <= next_BP_tx1_add4;
                BP_tx1_add3 <= next_BP_tx1_add3;
                BP_tx1_add2 <= next_BP_tx1_add2;
                BP_tx1_add1 <= next_BP_tx1_add1;
                BP_tx1_add0 <= next_BP_tx1_add0;
                BP_tx1_bus2 <= next_BP_tx1_bus2;
                BP_tx1_bus1 <= next_BP_tx1_bus1;
                BP_tx1_bus0 <= next_BP_tx1_bus0;
                bus_na2 <= next_bus_na2;
                bus_na1 <= next_bus_na1;
```

```
                bus_na0 <= next_bus_na0;
                tx1_next_bus2 <= next_tx1_next_bus2;
                tx1_next_bus1 <= next_tx1_next_bus1;
                tx1_next_bus0 <= next_tx1_next_bus0;
            END IF;
        END PROCESS;

        PROCESS (sreg,sreg1,BP_no_tx0,BP_no_tx1,BP_tx0_add0,BP_tx0_add1,BP_tx0_add2,

        BP_tx0_add3,BP_tx0_add4,BP_tx0_add5,BP_tx0_add6,BP_tx0_add7,BP_tx0_bus0,

        BP_tx0_bus1,BP_tx0_bus2,BP_tx0_eof,BP_tx0_sof,BP_tx0_src_rdy,BP_tx1_add0,

        BP_tx1_add1,BP_tx1_add2,BP_tx1_add3,BP_tx1_add4,BP_tx1_add5,BP_tx1_add6,
            BP_tx1_add7,BP_tx1_bus0,BP_tx1_bus1,BP_tx1_bus2,BP_tx1_eof,BP_tx1_sof,

        BP_tx1_src_rdy,bus0_na0,bus0_na1,bus0_na2,bus_na0,bus_na1,bus_na2,bus_status0

        ,bus_status1,bus_status2,tx0_add_en,tx0_new_bus0,tx0_new_bus1,tx0_new_bus2,
            tx0_next_bus0,tx0_next_bus1,tx0_next_bus2,tx1_add_en,tx1_new_bus0,
            tx1_new_bus1,tx1_new_bus2,tx1_next_bus0,tx1_next_bus1,tx1_next_bus2,
            BP_tx0_add,BP_tx0_bus,BP_tx1_add,BP_tx1_bus,bus0_na,bus_na,tx0_next_bus,
            tx1_next_bus)
        BEGIN
            next_BP_no_tx0 <= BP_no_tx0;next_BP_no_tx1 <=
BP_no_tx1;next_BP_tx0_add0 <=
                 BP_tx0_add0;next_BP_tx0_add1 <= BP_tx0_add1;next_BP_tx0_add2
<= BP_tx0_add2;
                next_BP_tx0_add3 <= BP_tx0_add3;next_BP_tx0_add4 <=
BP_tx0_add4;
                next_BP_tx0_add5 <= BP_tx0_add5;next_BP_tx0_add6 <=
BP_tx0_add6;
                next_BP_tx0_add7 <= BP_tx0_add7;next_BP_tx0_bus0 <= BP_tx0_bus0;
                next_BP_tx0_bus1 <= BP_tx0_bus1;next_BP_tx0_bus2 <= BP_tx0_bus2;
                next_BP_tx0_eof <= BP_tx0_eof;next_BP_tx0_sof <= BP_tx0_sof;
                next_BP_tx0_src_rdy <= BP_tx0_src_rdy;next_BP_tx1_add0 <=
BP_tx1_add0;
                next_BP_tx1_add1 <= BP_tx1_add1;next_BP_tx1_add2 <=
BP_tx1_add2;
                next_BP_tx1_add3 <= BP_tx1_add3;next_BP_tx1_add4 <=
BP_tx1_add4;
                next_BP_tx1_add5 <= BP_tx1_add5;next_BP_tx1_add6 <=
BP_tx1_add6;
                next_BP_tx1_add7 <= BP_tx1_add7;next_BP_tx1_bus0 <= BP_tx1_bus0;
                next_BP_tx1_bus1 <= BP_tx1_bus1;next_BP_tx1_bus2 <= BP_tx1_bus2;
                next_BP_tx1_eof <= BP_tx1_eof;next_BP_tx1_sof <= BP_tx1_sof;
```

```
                    next_BP_tx1_src_rdy <= BP_tx1_src_rdy;next_bus0_na0 <=
bus0_na0;next_bus0_na1
                     <= bus0_na1;next_bus0_na2 <= bus0_na2;next_bus_na0 <=
bus_na0;next_bus_na1
                    <= bus_na1;next_bus_na2 <= bus_na2;next_tx0_next_bus0 <=
tx0_next_bus0;
                    next_tx0_next_bus1 <= tx0_next_bus1;next_tx0_next_bus2 <=
tx0_next_bus2;
                    next_tx1_next_bus0 <= tx1_next_bus0;next_tx1_next_bus1 <=
tx1_next_bus1;
                    next_tx1_next_bus2 <= tx1_next_bus2;

            BP_tx0_add <= (( std_logic_vector'(BP_tx0_add7, BP_tx0_add6, BP_tx0_add5,
                    BP_tx0_add4, BP_tx0_add3, BP_tx0_add2, BP_tx0_add1,
BP_tx0_add0)));
            BP_tx0_bus <= (( std_logic_vector'(BP_tx0_bus2, BP_tx0_bus1, BP_tx0_bus0)))
                    ;
            BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7, BP_tx1_add6, BP_tx1_add5,
                    BP_tx1_add4, BP_tx1_add3, BP_tx1_add2, BP_tx1_add1,
BP_tx1_add0)));
            BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2, BP_tx1_bus1, BP_tx1_bus0)))
                    ;
            bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1, bus0_na0)));
            bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
            tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2, tx0_next_bus1,
                    tx0_next_bus0)));
            tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2, tx1_next_bus1,
                    tx1_next_bus0)));


            next_sreg<=END_TX0;
            next_sreg1<=END_TX1;

            CASE sreg IS
                    WHEN END_TX0 =>
                            next_sreg<=TX0_IDLE;
                            next_BP_tx0_sof<='1';
                            next_BP_tx0_eof<='1';
                            next_BP_no_tx0<='1';
                            next_BP_tx0_src_rdy<='1';

                            bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1, bus0_na0)));
                            tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                                    tx0_next_bus0)));
```

```
                        BP_tx0_add <= ( ( (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6,
                                BP_tx0_add5, BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)
                                ) +std_logic_vector'("00000001") )));
                        BP_tx0_bus <= (std_logic_vector'("000"));
                WHEN START_TX0 =>
                        next_sreg<=TX0_ON;
                        next_BP_tx0_sof<='0';

                        bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1, bus0_na0)));
                        BP_tx0_bus <= (( std_logic_vector'(BP_tx0_bus2, BP_tx0_bus1,
BP_tx0_bus0)
                                ));
                        tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                                tx0_next_bus0)));
                        IF (( BP_no_tx0='1' )) THEN next_BP_no_tx0<='1';
                        ELSE next_BP_no_tx0<='0';
                        END IF;

                        IF (( BP_tx0_eof='1' )) THEN next_BP_tx0_eof<='1';
                        ELSE next_BP_tx0_eof<='0';
                        END IF;

                        next_BP_tx0_src_rdy<='0';

                        BP_tx0_add <= ( ( (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6,
                                BP_tx0_add5, BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)
                                ) +std_logic_vector'("00000001") ) AND (
std_logic_vector'( tx0_add_en,
                                tx0_add_en, tx0_add_en, tx0_add_en, tx0_add_en,
tx0_add_en, tx0_add_en,
                                tx0_add_en)) ) OR  (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6, BP_tx0_add5
                                , BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)) AND (
                                std_logic_vector'( NOT tx0_add_en, NOT tx0_add_en,
NOT tx0_add_en, NOT
                                tx0_add_en, NOT tx0_add_en, NOT tx0_add_en, NOT
tx0_add_en, NOT tx0_add_en))
                                ));
                WHEN TX0_IDLE =>
```

```
                    IF ( tx0_new_bus0='1' ) OR ( tx0_new_bus1='1' ) OR (
tx0_new_bus2='1' )
                        THEN
                        next_sreg<=TX0_SETUP;
                        next_BP_no_tx0<='0';

                        BP_tx0_add <= (( std_logic_vector("00000000")));
                        BP_tx0_bus <= (( std_logic_vector(BP_tx0_bus2,
BP_tx0_bus1, BP_tx0_bus0
                            )));
                        IF (( BP_tx0_eof='1' )) THEN next_BP_tx0_eof<='1';
                        ELSE next_BP_tx0_eof<='0';
                        END IF;

                        IF (( BP_tx0_sof='1' )) THEN next_BP_tx0_sof<='1';
                        ELSE next_BP_tx0_sof<='0';
                        END IF;

                        IF (( BP_tx0_src_rdy='1' )) THEN
next_BP_tx0_src_rdy<='1';
                        ELSE next_BP_tx0_src_rdy<='0';
                        END IF;

                        tx0_next_bus <= (( std_logic_vector'(tx0_new_bus2,
tx0_new_bus1,
                            tx0_new_bus0)));
                        bus0_na <= (( std_logic_vector'(tx0_new_bus2,
tx0_new_bus1, tx0_new_bus0
                            )) AND ( std_logic_vector'(bus_status2,
bus_status1, bus_status0)));
                    ELSE
                        next_sreg<=TX0_IDLE;
                        next_BP_tx0_sof<='1';
                        next_BP_tx0_eof<='1';
                        next_BP_no_tx0<='1';
                        next_BP_tx0_src_rdy<='1';

                        bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1,
bus0_na0)));
                        tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                            tx0_next_bus0)));
                        BP_tx0_add <= (std_logic_vector("00000000"));
                        BP_tx0_bus <= (std_logic_vector("000"));
                    END IF;
                WHEN TX0_ON =>
```

```
                    IF ( BP_tx0_add0='1' AND BP_tx0_add1='1' AND
BP_tx0_add2='1' AND
                        BP_tx0_add3='1' AND BP_tx0_add4='1' AND
BP_tx0_add5='0' AND BP_tx0_add6='0'
                        AND BP_tx0_add7='0' ) THEN
                        next_sreg<=END_TX0;
                        next_BP_tx0_eof<='0';

                        bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1,
bus0_na0)));
                        BP_tx0_add <= (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6, BP_tx0_add5
                            , BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0))+
                            std_logic_vector'("00000001"));
                        BP_tx0_bus <= (( std_logic_vector'(BP_tx0_bus2,
BP_tx0_bus1, BP_tx0_bus0
                            )));
                        tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                            tx0_next_bus0)));
                        IF (( BP_no_tx0='1' )) THEN next_BP_no_tx0<='1';
                        ELSE next_BP_no_tx0<='0';
                        END IF;

                        IF (( BP_tx0_sof='1' )) THEN next_BP_tx0_sof<='1';
                        ELSE next_BP_tx0_sof<='0';
                        END IF;

                        IF (( BP_tx0_src_rdy='1' )) THEN
next_BP_tx0_src_rdy<='1';
                        ELSE next_BP_tx0_src_rdy<='0';
                        END IF;

                    ELSE
                        next_sreg<=TX0_ON;
                        next_BP_tx0_sof<='1';

                        bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1,
bus0_na0)));
                        BP_tx0_bus <= (( std_logic_vector'(BP_tx0_bus2,
BP_tx0_bus1, BP_tx0_bus0
                            )));
                        tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                            tx0_next_bus0)));
```

```
                                    IF (( BP_no_tx0='1' )) THEN next_BP_no_tx0<='1';
                                    ELSE next_BP_no_tx0<='0';
                                    END IF;

                                    IF (( BP_tx0_eof='1' )) THEN next_BP_tx0_eof<='1';
                                    ELSE next_BP_tx0_eof<='0';
                                    END IF;
                                    next_BP_tx0_src_rdy<='0';

                                    BP_tx0_add <= ( ( (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6,
                                        BP_tx0_add5, BP_tx0_add4, BP_tx0_add3,
BP_tx0_add2, BP_tx0_add1, BP_tx0_add0)
                                        ) +std_logic_vector("00000001") ) AND (
std_logic_vector'( tx0_add_en,
                                        tx0_add_en, tx0_add_en, tx0_add_en, tx0_add_en,
tx0_add_en, tx0_add_en,
                                        tx0_add_en)) ) OR  ((
std_logic_vector'(BP_tx0_add7, BP_tx0_add6, BP_tx0_add5
                                        , BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)) AND (
                                        std_logic_vector'( NOT tx0_add_en, NOT
tx0_add_en, NOT tx0_add_en, NOT
                                        tx0_add_en, NOT tx0_add_en, NOT tx0_add_en,
NOT tx0_add_en, NOT tx0_add_en))
                                        ));
                                END IF;
                        WHEN TX0_SETUP =>
                                IF ( bus0_na0='0' AND bus0_na1='0' AND bus0_na2='0' ) THEN
                                    next_sreg<=START_TX0;
                                    next_BP_tx0_sof<='1';
                                    next_BP_tx0_src_rdy<='1';

                                    bus0_na <= (( std_logic_vector'(bus0_na2, bus0_na1,
bus0_na0)));
                                    BP_tx0_add <= (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6, BP_tx0_add5
                                        , BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)));
                                    tx0_next_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                                        tx0_next_bus0)));
                                    IF (( BP_no_tx0='1' )) THEN next_BP_no_tx0<='1';
                                    ELSE next_BP_no_tx0<='0';
                                    END IF;
```

```
                                IF (( BP_tx0_eof='1' )) THEN next_BP_tx0_eof<='1';
                                ELSE next_BP_tx0_eof<='0';
                                END IF;

                                BP_tx0_bus <= (( std_logic_vector'(tx0_next_bus2,
tx0_next_bus1,
                                        tx0_next_bus0)));
                        ELSE
                                next_sreg<=TX0_SETUP;
                                next_BP_no_tx0<='0';

                                BP_tx0_add <= (( std_logic_vector'(BP_tx0_add7,
BP_tx0_add6, BP_tx0_add5
                                        , BP_tx0_add4, BP_tx0_add3, BP_tx0_add2,
BP_tx0_add1, BP_tx0_add0)));
                                BP_tx0_bus <= (( std_logic_vector'(BP_tx0_bus2,
BP_tx0_bus1, BP_tx0_bus0
                                        )));
                                IF (( BP_tx0_eof='1' )) THEN next_BP_tx0_eof<='1';
                                ELSE next_BP_tx0_eof<='0';
                                END IF;

                                IF (( BP_tx0_sof='1' )) THEN next_BP_tx0_sof<='1';
                                ELSE next_BP_tx0_sof<='0';
                                END IF;

                                IF (( BP_tx0_src_rdy='1' )) THEN
next_BP_tx0_src_rdy<='1';
                                ELSE next_BP_tx0_src_rdy<='0';
                                END IF;

                                tx0_next_bus <= (( std_logic_vector'(tx0_new_bus2,
tx0_new_bus1,
                                        tx0_new_bus0)));
                                bus0_na <= (( std_logic_vector'(tx0_new_bus2,
tx0_new_bus1, tx0_new_bus0
                                        )) AND ( std_logic_vector'(bus_status2,
bus_status1, bus_status0)));
                        END IF;
                WHEN OTHERS =>
        END CASE;

        CASE sreg1 IS
                WHEN END_TX1 =>
                        next_sreg1<=TX1_IDLE;
                        next_BP_tx1_sof<='1';
```

```
                        next_BP_tx1_eof<='1';
                        next_BP_no_tx1<='1';
                        next_BP_tx1_src_rdy<='1';

                        bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
                        tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                tx1_next_bus0)));
                        BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7, BP_tx1_add6,
                                BP_tx1_add5, BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)
                                ) +std_logic_vector'("00000001"));
                        BP_tx1_bus <= (std_logic_vector'("000"));
                WHEN START_TX1 =>
                        next_sreg1<=TX1_ON;
                        next_BP_tx1_sof<='0';

                        bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
                        BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2, BP_tx1_bus1,
BP_tx1_bus0)
                                ));
                        tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                tx1_next_bus0)));
                        IF (( BP_no_tx1='1' )) THEN next_BP_no_tx1<='1';
                        ELSE next_BP_no_tx1<='0';
                        END IF;

                        IF (( BP_tx1_eof='1' )) THEN next_BP_tx1_eof<='1';
                        ELSE next_BP_tx1_eof<='0';
                        END IF;

                        next_BP_tx1_src_rdy<='0';

                        BP_tx1_add <= ( ( (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6,
                                BP_tx1_add5, BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)
                                ) +std_logic_vector'("00000001") ) AND (
std_logic_vector'( tx1_add_en,
                                tx1_add_en, tx1_add_en, tx1_add_en, tx1_add_en,
tx1_add_en, tx1_add_en,
                                tx1_add_en)) ) OR  (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6, BP_tx1_add5
                                , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)) AND (
```

```
                                    std_logic_vector'( NOT tx1_add_en, NOT tx1_add_en,
NOT tx1_add_en, NOT
                                    tx1_add_en, NOT tx1_add_en, NOT tx1_add_en, NOT
tx1_add_en, NOT tx1_add_en))
                                    ));
                        WHEN TX1_IDLE =>
                                IF ( tx1_new_bus0='1' ) OR ( tx1_new_bus1='1' ) OR (
tx1_new_bus2='1' )
                                    THEN
                                    next_sreg1<=TX1_SETUP;
                                    next_BP_no_tx1<='0';

                                    BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6, BP_tx1_add5
                                        , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)));
                                    BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2,
BP_tx1_bus1, BP_tx1_bus0
                                        )));
                                    IF (( BP_tx1_eof='1' )) THEN next_BP_tx1_eof<='1';
                                    ELSE next_BP_tx1_eof<='0';
                                    END IF;

                                    IF (( BP_tx1_sof='1' )) THEN next_BP_tx1_sof<='1';
                                    ELSE next_BP_tx1_sof<='0';
                                    END IF;

                                    IF (( BP_tx1_src_rdy='1' )) THEN
next_BP_tx1_src_rdy<='1';
                                    ELSE next_BP_tx1_src_rdy<='0';
                                    END IF;

                                    tx1_next_bus <= (( std_logic_vector'(tx1_new_bus2,
tx1_new_bus1,
                                        tx1_new_bus0)));
                                    bus_na <= (( std_logic_vector'(tx1_new_bus2,
tx1_new_bus1, tx1_new_bus0)
                                        ) AND ( std_logic_vector'(bus_status2, bus_status1,
bus_status0)));
                                ELSE
                                    next_sreg1<=TX1_IDLE;
                                    next_BP_tx1_sof<='1';
                                    next_BP_tx1_eof<='1';
                                    next_BP_no_tx1<='1';
                                    next_BP_tx1_src_rdy<='1';
```

```
                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                        tx1_next_bus0)));
                                    BP_tx1_add <= (std_logic_vector'("00100000"));
                                    BP_tx1_bus <= (std_logic_vector'("000"));
                                END IF;
                            WHEN TX1_ON =>
                                IF ( BP_tx1_add0='1' AND BP_tx1_add1='1' AND
BP_tx1_add2='1' AND
                                    BP_tx1_add3='1' AND BP_tx1_add4='1' AND
BP_tx1_add5='1' AND BP_tx1_add6='0'
                                    AND BP_tx1_add7='0' ) THEN
                                    next_sreg1<=END_TX1;
                                    next_BP_tx1_eof<='0';

                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6, BP_tx1_add5
                                        , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0))+
                                        std_logic_vector'("00000001"));
                                    BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2,
BP_tx1_bus1, BP_tx1_bus0
                                        )));
                                    tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                        tx1_next_bus0)));
                                    IF (( BP_no_tx1='1' )) THEN next_BP_no_tx1<='1';
                                    ELSE next_BP_no_tx1<='0';
                                    END IF;

                                    IF (( BP_tx1_sof='1' )) THEN next_BP_tx1_sof<='1';
                                    ELSE next_BP_tx1_sof<='0';
                                    END IF;

                                    IF (( BP_tx1_src_rdy='1' )) THEN
next_BP_tx1_src_rdy<='1';
                                    ELSE next_BP_tx1_src_rdy<='0';
                                    END IF;

                                ELSE
                                    next_sreg1<=TX1_ON;
                                    next_BP_tx1_sof<='1';
```

```
                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2,
BP_tx1_bus1, BP_tx1_bus0
                                        )));
                                    tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                        tx1_next_bus0)));
                                    IF (( BP_no_tx1='1' )) THEN next_BP_no_tx1<='1';
                                    ELSE next_BP_no_tx1<='0';
                                    END IF;

                                    IF (( BP_tx1_eof='1' )) THEN next_BP_tx1_eof<='1';
                                    ELSE next_BP_tx1_eof<='0';
                                    END IF;

                                    IF (( BP_tx1_src_rdy='1' )) THEN
next_BP_tx1_src_rdy<='1';
                                    ELSE next_BP_tx1_src_rdy<='0';
                                    END IF;

                                    BP_tx1_add <= ( ( (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6,
                                        BP_tx1_add5, BP_tx1_add4, BP_tx1_add3,
BP_tx1_add2, BP_tx1_add1, BP_tx1_add0)
                                        ) +std_logic_vector'("00000001") ) AND (
std_logic_vector'( tx1_add_en,
                                        tx1_add_en, tx1_add_en, tx1_add_en, tx1_add_en,
tx1_add_en, tx1_add_en,
                                        tx1_add_en)) ) OR  ((
std_logic_vector'(BP_tx1_add7, BP_tx1_add6, BP_tx1_add5
                                        , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)) AND (
                                        std_logic_vector'( NOT tx1_add_en, NOT
tx1_add_en, NOT tx1_add_en, NOT
                                        tx1_add_en, NOT tx1_add_en, NOT tx1_add_en,
NOT tx1_add_en, NOT tx1_add_en))
                                        ));
                                END IF;
                            WHEN TX1_SETUP =>
                                IF ( bus_na0='0' AND bus_na1='0' AND bus_na2='0' ) THEN
                                    next_sreg1<=START_TX1;
                                    next_BP_tx1_sof<='1';
                                    next_BP_tx1_src_rdy<='1';
```

```
                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6, BP_tx1_add5
                                        , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)));
                                    tx1_next_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                        tx1_next_bus0)));
                                    IF (( BP_no_tx1='1' )) THEN next_BP_no_tx1<='1';
                                    ELSE next_BP_no_tx1<='0';
                                    END IF;

                                    IF (( BP_tx1_eof='1' )) THEN next_BP_tx1_eof<='1';
                                    ELSE next_BP_tx1_eof<='0';
                                    END IF;

                                    BP_tx1_bus <= (( std_logic_vector'(tx1_next_bus2,
tx1_next_bus1,
                                        tx1_next_bus0)));
                                ELSE
                                    next_sreg1<=TX1_SETUP;
                                    next_BP_no_tx1<='0';

                                    BP_tx1_add <= (( std_logic_vector'(BP_tx1_add7,
BP_tx1_add6, BP_tx1_add5
                                        , BP_tx1_add4, BP_tx1_add3, BP_tx1_add2,
BP_tx1_add1, BP_tx1_add0)));
                                    BP_tx1_bus <= (( std_logic_vector'(BP_tx1_bus2,
BP_tx1_bus1, BP_tx1_bus0
                                        )));
                                    IF (( BP_tx1_eof='1' )) THEN next_BP_tx1_eof<='1';
                                    ELSE next_BP_tx1_eof<='0';
                                    END IF;

                                    IF (( BP_tx1_sof='1' )) THEN next_BP_tx1_sof<='1';
                                    ELSE next_BP_tx1_sof<='0';
                                    END IF;

                                    IF (( BP_tx1_src_rdy='1' )) THEN
next_BP_tx1_src_rdy<='1';
                                    ELSE next_BP_tx1_src_rdy<='0';
                                    END IF;

                                    tx1_next_bus <= (( std_logic_vector'(tx1_new_bus2,
tx1_new_bus1,
```

```
                                        tx1_new_bus0)));
                                    bus_na <= (( std_logic_vector'(tx1_new_bus2,
tx1_new_bus1, tx1_new_bus0)
                                        ) AND ( std_logic_vector'(bus_status2, bus_status1,
bus_status0)));
                                END IF;
                            WHEN OTHERS =>
                    END CASE;

                    next_BP_tx0_add7 <= BP_tx0_add(7);
                    next_BP_tx0_add6 <= BP_tx0_add(6);
                    next_BP_tx0_add5 <= BP_tx0_add(5);
                    next_BP_tx0_add4 <= BP_tx0_add(4);
                    next_BP_tx0_add3 <= BP_tx0_add(3);
                    next_BP_tx0_add2 <= BP_tx0_add(2);
                    next_BP_tx0_add1 <= BP_tx0_add(1);
                    next_BP_tx0_add0 <= BP_tx0_add(0);
                    next_BP_tx0_bus2 <= BP_tx0_bus(2);
                    next_BP_tx0_bus1 <= BP_tx0_bus(1);
                    next_BP_tx0_bus0 <= BP_tx0_bus(0);
                    next_BP_tx1_add7 <= BP_tx1_add(7);
                    next_BP_tx1_add6 <= BP_tx1_add(6);
                    next_BP_tx1_add5 <= BP_tx1_add(5);
                    next_BP_tx1_add4 <= BP_tx1_add(4);
                    next_BP_tx1_add3 <= BP_tx1_add(3);
                    next_BP_tx1_add2 <= BP_tx1_add(2);
                    next_BP_tx1_add1 <= BP_tx1_add(1);
                    next_BP_tx1_add0 <= BP_tx1_add(0);
                    next_BP_tx1_bus2 <= BP_tx1_bus(2);
                    next_BP_tx1_bus1 <= BP_tx1_bus(1);
                    next_BP_tx1_bus0 <= BP_tx1_bus(0);
                    next_bus0_na2 <= bus0_na(2);
                    next_bus0_na1 <= bus0_na(1);
                    next_bus0_na0 <= bus0_na(0);
                    next_bus_na2 <= bus_na(2);
                    next_bus_na1 <= bus_na(1);
                    next_bus_na0 <= bus_na(0);
                    next_tx0_next_bus2 <= tx0_next_bus(2);
                    next_tx0_next_bus1 <= tx0_next_bus(1);
                    next_tx0_next_bus0 <= tx0_next_bus(0);
                    next_tx1_next_bus2 <= tx1_next_bus(2);
                    next_tx1_next_bus1 <= tx1_next_bus(1);
                    next_tx1_next_bus0 <= tx1_next_bus(0);
        END PROCESS;

        PROCESS (BP_no_tx0)
```

```
BEGIN
        IF (( BP_no_tx0='1' )) THEN no_tx0<='1';
        ELSE no_tx0<='0';
        END IF;
END PROCESS;

PROCESS (BP_no_tx1)
BEGIN
        IF (( BP_no_tx1='1' )) THEN no_tx1<='1';
        ELSE no_tx1<='0';
        END IF;
END PROCESS;

PROCESS (BP_tx0_eof)
BEGIN
        IF (( BP_tx0_eof='1' )) THEN tx0_eof<='1';
        ELSE tx0_eof<='0';
        END IF;
END PROCESS;

PROCESS (BP_tx0_sof)
BEGIN
        IF (( BP_tx0_sof='1' )) THEN tx0_sof<='1';
        ELSE tx0_sof<='0';
        END IF;
END PROCESS;

PROCESS (BP_tx0_src_rdy)
BEGIN
        IF (( BP_tx0_src_rdy='1' )) THEN tx0_src_rdy<='1';
        ELSE tx0_src_rdy<='0';
        END IF;
END PROCESS;

PROCESS (BP_tx1_eof)
BEGIN
        IF (( BP_tx1_eof='1' )) THEN tx1_eof<='1';
        ELSE tx1_eof<='0';
        END IF;
END PROCESS;

PROCESS (BP_tx1_sof)
BEGIN
        IF (( BP_tx1_sof='1' )) THEN tx1_sof<='1';
        ELSE tx1_sof<='0';
        END IF;
```

```
	END PROCESS;

	PROCESS (BP_tx1_src_rdy)
	BEGIN
		IF (( BP_tx1_src_rdy='1' )) THEN tx1_src_rdy<='1';
		ELSE tx1_src_rdy<='0';
		END IF;
	END PROCESS;

	PROCESS (BP_tx0_add0,BP_tx0_add1,BP_tx0_add2,BP_tx0_add3,BP_tx0_add4,
		BP_tx0_add5,BP_tx0_add6,BP_tx0_add7,tx0_add)
	BEGIN
		tx0_add <= (( std_logic_vector'(BP_tx0_add7, BP_tx0_add6, BP_tx0_add5,
			BP_tx0_add4, BP_tx0_add3, BP_tx0_add2, BP_tx0_add1,
BP_tx0_add0)));
		tx0_add0 <= tx0_add(0);
		tx0_add1 <= tx0_add(1);
		tx0_add2 <= tx0_add(2);
		tx0_add3 <= tx0_add(3);
		tx0_add4 <= tx0_add(4);
		tx0_add5 <= tx0_add(5);
		tx0_add6 <= tx0_add(6);
		tx0_add7 <= tx0_add(7);
	END PROCESS;

	PROCESS (BP_tx0_bus0,BP_tx0_bus1,BP_tx0_bus2,tx0_bus)
	BEGIN
		tx0_bus <= (( std_logic_vector'(BP_tx0_bus2, BP_tx0_bus1, BP_tx0_bus0)));
		tx0_bus0 <= tx0_bus(0);
		tx0_bus1 <= tx0_bus(1);
		tx0_bus2 <= tx0_bus(2);
	END PROCESS;

	PROCESS (BP_tx1_add0,BP_tx1_add1,BP_tx1_add2,BP_tx1_add3,BP_tx1_add4,
		BP_tx1_add5,BP_tx1_add6,BP_tx1_add7,tx1_add)
	BEGIN
		tx1_add <= (( std_logic_vector'(BP_tx1_add7, BP_tx1_add6, BP_tx1_add5,
			BP_tx1_add4, BP_tx1_add3, BP_tx1_add2, BP_tx1_add1,
BP_tx1_add0)));
		tx1_add0 <= tx1_add(0);
		tx1_add1 <= tx1_add(1);
		tx1_add2 <= tx1_add(2);
		tx1_add3 <= tx1_add(3);
		tx1_add4 <= tx1_add(4);
		tx1_add5 <= tx1_add(5);
		tx1_add6 <= tx1_add(6);
```

```
		tx1_add7 <= tx1_add(7);
	END PROCESS;

	PROCESS (BP_tx1_bus0,BP_tx1_bus1,BP_tx1_bus2,tx1_bus)
	BEGIN
		tx1_bus <= (( std_logic_vector'(BP_tx1_bus2, BP_tx1_bus1, BP_tx1_bus0)));
		tx1_bus0 <= tx1_bus(0);
		tx1_bus1 <= tx1_bus(1);
		tx1_bus2 <= tx1_bus(2);
	END PROCESS;
END BEHAVIOR;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY TRANS IS
	PORT (bus_status : IN std_logic_vector (2 DOWNTO 0);
		tx0_add : OUT std_logic_vector (7 DOWNTO 0);
		tx0_bus : OUT std_logic_vector (2 DOWNTO 0);
		tx0_dat : IN std_logic_vector (7 DOWNTO 0);
		tx0_new_bus : IN std_logic_vector (2 DOWNTO 0);
		tx1_add : OUT std_logic_vector (7 DOWNTO 0);
		tx1_bus : OUT std_logic_vector (2 DOWNTO 0);
		tx1_dat : IN std_logic_vector (7 DOWNTO 0);
		tx1_new_bus : IN std_logic_vector (2 DOWNTO 0);
		CLK,RESET,tx0_add_en,tx1_add_en: IN std_logic;
		no_tx0,no_tx1,tx0_eof,tx0_sof,tx0_src_rdy,tx1_eof,tx1_sof,tx1_src_rdy : OUT
			std_logic);
END;

ARCHITECTURE BEHAVIOR OF TRANS IS
	COMPONENT SHELL_TRANS
		PORT (CLK,bus_status0,bus_status1,bus_status2,RESET,tx0_add_en,tx0_dat0,

	tx0_dat1,tx0_dat2,tx0_dat3,tx0_dat4,tx0_dat5,tx0_dat6,tx0_dat7,tx0_new_bus0,

	tx0_new_bus1,tx0_new_bus2,tx1_add_en,tx1_dat0,tx1_dat1,tx1_dat2,tx1_dat3,

	tx1_dat4,tx1_dat5,tx1_dat6,tx1_dat7,tx1_new_bus0,tx1_new_bus1,tx1_new_bus2:
			IN std_logic;
			no_tx0,no_tx1,tx0_add0,tx0_add1,tx0_add2,tx0_add3,tx0_add4,tx0_add5,

	tx0_add6,tx0_add7,tx0_bus0,tx0_bus1,tx0_bus2,tx0_eof,tx0_sof,tx0_src_rdy,
```

```
        tx1_add0,tx1_add1,tx1_add2,tx1_add3,tx1_add4,tx1_add5,tx1_add6,tx1_add7,
                                tx1_bus0,tx1_bus1,tx1_bus2,tx1_eof,tx1_sof,tx1_src_rdy : OUT
std_logic);
        END COMPONENT;
BEGIN
        SHELL1_TRANS : SHELL_TRANS PORT MAP
(CLK=>CLK,bus_status0=>bus_status(0),

        bus_status1=>bus_status(1),bus_status2=>bus_status(2),RESET=>RESET,tx0_add_en

        =>tx0_add_en,tx0_dat0=>tx0_dat(0),tx0_dat1=>tx0_dat(1),tx0_dat2=>tx0_dat(2),
                tx0_dat3=>tx0_dat(3),tx0_dat4=>tx0_dat(4),tx0_dat5=>tx0_dat(5),tx0_dat6=>

        tx0_dat(6),tx0_dat7=>tx0_dat(7),tx0_new_bus0=>tx0_new_bus(0),tx0_new_bus1=>

        tx0_new_bus(1),tx0_new_bus2=>tx0_new_bus(2),tx1_add_en=>tx1_add_en,tx1_dat0=>
                tx1_dat(0),tx1_dat1=>tx1_dat(1),tx1_dat2=>tx1_dat(2),tx1_dat3=>tx1_dat(3),
                tx1_dat4=>tx1_dat(4),tx1_dat5=>tx1_dat(5),tx1_dat6=>tx1_dat(6),tx1_dat7=>
                tx1_dat(7),tx1_new_bus0=>tx1_new_bus(0),tx1_new_bus1=>tx1_new_bus(1),

        tx1_new_bus2=>tx1_new_bus(2),no_tx0=>no_tx0,no_tx1=>no_tx1,tx0_add0=>tx0_add(

        0),tx0_add1=>tx0_add(1),tx0_add2=>tx0_add(2),tx0_add3=>tx0_add(3),tx0_add4=>

        tx0_add(4),tx0_add5=>tx0_add(5),tx0_add6=>tx0_add(6),tx0_add7=>tx0_add(7),
                tx0_bus0=>tx0_bus(0),tx0_bus1=>tx0_bus(1),tx0_bus2=>tx0_bus(2),tx0_eof=>
                tx0_eof,tx0_sof=>tx0_sof,tx0_src_rdy=>tx0_src_rdy,tx1_add0=>tx1_add(0),

        tx1_add1=>tx1_add(1),tx1_add2=>tx1_add(2),tx1_add3=>tx1_add(3),tx1_add4=>

        tx1_add(4),tx1_add5=>tx1_add(5),tx1_add6=>tx1_add(6),tx1_add7=>tx1_add(7),
                tx1_bus0=>tx1_bus(0),tx1_bus1=>tx1_bus(1),tx1_bus2=>tx1_bus(2),tx1_eof=>
                tx1_eof,tx1_sof=>tx1_sof,tx1_src_rdy=>tx1_src_rdy);
END BEHAVIOR;
```

Interface State Machine:

```
--  C:\XILINX\TRIDENT\INTERFACE\INTF.vhd
--  VHDL code created by Xilinx's StateCAD 6.2i
--  Thu Apr 14 16:30:56 2005

--  This VHDL code (for use with IEEE compliant tools) was generated using:
--  enumerated state assignment with structured code format.
--  Minimization is enabled,  implied else is enabled,
--  and outputs are speed optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY SHELL_INTF IS
        PORT
(CLK,bus_status0,bus_status1,bus_status2,f_dat0,f_dat1,f_dat2,f_dat3,f_dat4,f_dat5,f_dat6,f_da
t7
                ,RESET,rx0_sof,t_add_en,t_dat0,t_dat1,t_dat2,t_dat3,t_dat4,t_dat5,t_dat6,
                t_dat7,t_new_bus0,t_new_bus1,t_new_bus2,bus0_busy,bus1_busy,bus2_busy: IN
std_logic;
                f_add0,f_add1,f_add2,f_add3,f_add4,f_add5,f_add6,f_add7,f_cs0,f_cs1,f_cs2,

        no_t,t_add0,t_add1,t_add2,t_add3,t_add4,t_add5,t_add6,t_add7,t_bus0,t_bus1,
                        t_bus2,t_eof,t_sof,t_src_rdy : OUT std_logic);

        SIGNAL
BP_f_add0,BP_f_add1,BP_f_add2,BP_f_add3,BP_f_add4,BP_f_add5,BP_f_add6

        ,BP_f_add7,BP_f_cs0,BP_f_cs1,BP_f_cs2,BP_no_t,BP_t_add0,BP_t_add1,BP_t_add2,
                BP_t_add3,BP_t_add4,BP_t_add5,BP_t_add6,BP_t_add7,BP_t_bus0,BP_t_bus1,
                BP_t_bus2,BP_t_eof,BP_t_sof,BP_t_src_rdy,BP_tx0_mask0,BP_tx0_mask1,

        BP_tx0_mask2,BP_tx0_mask3,BP_tx0_mask4,BP_tx0_mask5,BP_tx0_mask6,BP_tx0_
mask7
                ,BP_tx1_mask0,BP_tx1_mask1,BP_tx1_mask2,BP_tx1_mask3,BP_tx1_mask4,

        BP_tx1_mask5,BP_tx1_mask6,BP_tx1_mask7,bus_na0,bus_na1,bus_na2,dest_tx00,
                dest_tx01,dest_tx02,dest_tx03,dest_tx04,dest_tx05,dest_tx06,dest_tx07,
                dest_tx10,dest_tx11,dest_tx12,dest_tx13,dest_tx14,dest_tx15,dest_tx16,
                dest_tx17,t_next_bus0,t_next_bus1,t_next_bus2: std_logic;
END;
```

```
ARCHITECTURE BEHAVIOR OF SHELL_INTF IS
	TYPE type_sreg IS (END_T,START_T,T_IDLE,T_ON,T_SETUP);
	SIGNAL sreg, next_sreg : type_sreg;
	TYPE type_sreg1 IS (f_analyze,F_IDLE,F_ON);
	SIGNAL sreg1, next_sreg1 : type_sreg1;
	SIGNAL next_BP_f_add0,next_BP_f_add1,next_BP_f_add2,next_BP_f_add3,

	next_BP_f_add4,next_BP_f_add5,next_BP_f_add6,next_BP_f_add7,next_BP_f_cs0,
		next_BP_f_cs1,next_BP_f_cs2,next_BP_no_t,next_BP_t_add0,next_BP_t_add1,

	next_BP_t_add2,next_BP_t_add3,next_BP_t_add4,next_BP_t_add5,next_BP_t_add6,

	next_BP_t_add7,next_BP_t_bus0,next_BP_t_bus1,next_BP_t_bus2,next_BP_t_eof,
		next_BP_t_sof,next_BP_t_src_rdy,next_bus_na0,next_bus_na1,next_bus_na2,
		next_dest_tx00,next_dest_tx01,next_dest_tx02,next_dest_tx03,next_dest_tx04,
		next_dest_tx05,next_dest_tx06,next_dest_tx07,next_dest_tx10,next_dest_tx11,
		next_dest_tx12,next_dest_tx13,next_dest_tx14,next_dest_tx15,next_dest_tx16,
		next_dest_tx17,next_t_next_bus0,next_t_next_bus1,next_t_next_bus2 :
		std_logic;
	SIGNAL BP_f_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL BP_f_cs : std_logic_vector (2 DOWNTO 0);
	SIGNAL BP_t_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL BP_t_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL BP_tx0_mask : std_logic_vector (7 DOWNTO 0);
	SIGNAL BP_tx1_mask : std_logic_vector (7 DOWNTO 0);
	SIGNAL bus_na : std_logic_vector (2 DOWNTO 0);
	SIGNAL dest_tx0 : std_logic_vector (7 DOWNTO 0);
	SIGNAL dest_tx1 : std_logic_vector (7 DOWNTO 0);
	SIGNAL f_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL f_cs : std_logic_vector (2 DOWNTO 0);
	SIGNAL t_add : std_logic_vector (7 DOWNTO 0);
	SIGNAL t_bus : std_logic_vector (2 DOWNTO 0);
	SIGNAL t_next_bus : std_logic_vector (2 DOWNTO 0);
BEGIN
	PROCESS (CLK, RESET, next_sreg, next_BP_no_t, next_BP_t_eof, next_BP_t_sof,
		next_BP_t_src_rdy, next_BP_t_add7, next_BP_t_add6, next_BP_t_add5,
		next_BP_t_add4, next_BP_t_add3, next_BP_t_add2, next_BP_t_add1,
		next_BP_t_add0, next_BP_t_bus2, next_BP_t_bus1, next_BP_t_bus0,
next_bus_na2,
		 next_bus_na1, next_bus_na0, next_t_next_bus2, next_t_next_bus1,
		next_t_next_bus0)
	BEGIN
		IF ( RESET='1' ) THEN
			sreg <= T_IDLE;
			bus_na2 <= '0';
			bus_na1 <= '0';
```

```
                bus_na0 <= '0';
                t_next_bus2 <= '0';
                t_next_bus1 <= '0';
                t_next_bus0 <= '0';
                BP_no_t <= '1';
                BP_t_eof <= '1';
                BP_t_sof <= '1';
                BP_t_src_rdy <= '1';
                BP_t_add7 <= '0';
                BP_t_add6 <= '1';
                BP_t_add5 <= '0';
                BP_t_add4 <= '0';
                BP_t_add3 <= '0';
                BP_t_add2 <= '0';
                BP_t_add1 <= '0';
                BP_t_add0 <= '0';
                BP_t_bus2 <= '0';
                BP_t_bus1 <= '0';
                BP_t_bus0 <= '0';
        ELSIF CLK='1' AND CLK'event THEN
                sreg <= next_sreg;
                BP_no_t <= next_BP_no_t;
                BP_t_eof <= next_BP_t_eof;
                BP_t_sof <= next_BP_t_sof;
                BP_t_src_rdy <= next_BP_t_src_rdy;
                BP_t_add7 <= next_BP_t_add7;
                BP_t_add6 <= next_BP_t_add6;
                BP_t_add5 <= next_BP_t_add5;
                BP_t_add4 <= next_BP_t_add4;
                BP_t_add3 <= next_BP_t_add3;
                BP_t_add2 <= next_BP_t_add2;
                BP_t_add1 <= next_BP_t_add1;
                BP_t_add0 <= next_BP_t_add0;
                BP_t_bus2 <= next_BP_t_bus2;
                BP_t_bus1 <= next_BP_t_bus1;
                BP_t_bus0 <= next_BP_t_bus0;
                bus_na2 <= next_bus_na2;
                bus_na1 <= next_bus_na1;
                bus_na0 <= next_bus_na0;
                t_next_bus2 <= next_t_next_bus2;
                t_next_bus1 <= next_t_next_bus1;
                t_next_bus0 <= next_t_next_bus0;
        END IF;
END PROCESS;

PROCESS (CLK, RESET, next_sreg1, next_BP_f_add7, next_BP_f_add6,
```

```
            next_BP_f_add5, next_BP_f_add4, next_BP_f_add3, next_BP_f_add2,
            next_BP_f_add1, next_BP_f_add0, next_BP_f_cs2, next_BP_f_cs1,
next_BP_f_cs0,
            next_dest_tx07, next_dest_tx06, next_dest_tx05, next_dest_tx04,
            next_dest_tx03, next_dest_tx02, next_dest_tx01, next_dest_tx00,
            next_dest_tx17, next_dest_tx16, next_dest_tx15, next_dest_tx14,
            next_dest_tx13, next_dest_tx12, next_dest_tx11, next_dest_tx10)
        BEGIN
            IF ( RESET='1' ) THEN
                sreg1 <= F_IDLE;
                BP_f_cs2 <= '0';
                BP_f_cs1 <= '0';
                BP_f_cs0 <= '0';
                dest_tx07 <= '0';
                dest_tx06 <= '0';
                dest_tx05 <= '0';
                dest_tx04 <= '0';
                dest_tx03 <= '0';
                dest_tx02 <= '0';
                dest_tx01 <= '0';
                dest_tx00 <= '0';
                dest_tx17 <= '0';
                dest_tx16 <= '0';
                dest_tx15 <= '0';
                dest_tx14 <= '0';
                dest_tx13 <= '0';
                dest_tx12 <= '0';
                dest_tx11 <= '0';
                dest_tx10 <= '0';
                BP_f_add7 <= '0';
                BP_f_add6 <= '0';
                BP_f_add5 <= '0';
                BP_f_add4 <= '0';
                BP_f_add3 <= '0';
                BP_f_add2 <= '0';
                BP_f_add1 <= '0';
                BP_f_add0 <= '0';
            ELSIF CLK='1' AND CLK'event THEN
                sreg1 <= next_sreg1;
                BP_f_add7 <= next_BP_f_add7;
                BP_f_add6 <= next_BP_f_add6;
                BP_f_add5 <= next_BP_f_add5;
                BP_f_add4 <= next_BP_f_add4;
                BP_f_add3 <= next_BP_f_add3;
                BP_f_add2 <= next_BP_f_add2;
                BP_f_add1 <= next_BP_f_add1;
```

```
                BP_f_add0 <= next_BP_f_add0;
                BP_f_cs2 <= next_BP_f_cs2;
                BP_f_cs1 <= next_BP_f_cs1;
                BP_f_cs0 <= next_BP_f_cs0;
                dest_tx07 <= next_dest_tx07;
                dest_tx06 <= next_dest_tx06;
                dest_tx05 <= next_dest_tx05;
                dest_tx04 <= next_dest_tx04;
                dest_tx03 <= next_dest_tx03;
                dest_tx02 <= next_dest_tx02;
                dest_tx01 <= next_dest_tx01;
                dest_tx00 <= next_dest_tx00;
                dest_tx17 <= next_dest_tx17;
                dest_tx16 <= next_dest_tx16;
                dest_tx15 <= next_dest_tx15;
                dest_tx14 <= next_dest_tx14;
                dest_tx13 <= next_dest_tx13;
                dest_tx12 <= next_dest_tx12;
                dest_tx11 <= next_dest_tx11;
                dest_tx10 <= next_dest_tx10;
        END IF;
    END PROCESS;

    PROCESS (sreg,sreg1,BP_f_add0,BP_f_add1,BP_f_add2,BP_f_add3,BP_f_add4,

    BP_f_add5,BP_f_add6,BP_f_add7,BP_f_cs0,BP_f_cs1,BP_f_cs2,BP_no_t,BP_t_add0,
        BP_t_add1,BP_t_add2,BP_t_add3,BP_t_add4,BP_t_add5,BP_t_add6,BP_t_add7,

    BP_t_bus0,BP_t_bus1,BP_t_bus2,BP_t_eof,BP_t_sof,BP_t_src_rdy,BP_tx0_mask0,

    BP_tx0_mask1,BP_tx0_mask2,BP_tx0_mask3,BP_tx0_mask4,BP_tx0_mask5,BP_tx0_
mask6
        ,BP_tx0_mask7,BP_tx1_mask0,BP_tx1_mask1,BP_tx1_mask2,BP_tx1_mask3,

    BP_tx1_mask4,BP_tx1_mask5,BP_tx1_mask6,BP_tx1_mask7,bus_na0,bus_na1,bus_na2
,

    bus_status0,bus_status1,bus_status2,dest_tx00,dest_tx01,dest_tx02,dest_tx03,dest_tx04,d
est_tx05,
        dest_tx06,dest_tx07,dest_tx10,dest_tx11,dest_tx12,dest_tx13,dest_tx14,
        dest_tx15,dest_tx16,dest_tx17,f_dat0,f_dat1,f_dat2,f_dat3,f_dat4,f_dat5,

    f_dat6,f_dat7,rx0_sof,t_add_en,t_new_bus0,t_new_bus1,t_new_bus2,t_next_bus0,

    t_next_bus1,t_next_bus2,BP_f_add,BP_f_cs,BP_t_add,BP_t_bus,bus_na,dest_tx0,
        dest_tx1,t_next_bus,bus0_busy,bus1_busy,bus2_busy)
```

```
	BEGIN
		next_BP_f_add0 <= BP_f_add0;next_BP_f_add1 <= BP_f_add1;next_BP_f_add2
<=
			BP_f_add2;next_BP_f_add3 <= BP_f_add3;next_BP_f_add4 <=
BP_f_add4;
			next_BP_f_add5 <= BP_f_add5;next_BP_f_add6 <=
BP_f_add6;next_BP_f_add7 <=
			BP_f_add7;next_BP_f_cs0 <= BP_f_cs0;next_BP_f_cs1 <=
BP_f_cs1;next_BP_f_cs2
			<= BP_f_cs2;next_BP_no_t <= BP_no_t;next_BP_t_add0 <= BP_t_add0;
			next_BP_t_add1 <= BP_t_add1;next_BP_t_add2 <=
BP_t_add2;next_BP_t_add3 <=
			BP_t_add3;next_BP_t_add4 <= BP_t_add4;next_BP_t_add5 <=
BP_t_add5;
			next_BP_t_add6 <= BP_t_add6;next_BP_t_add7 <=
BP_t_add7;next_BP_t_bus0 <=
			BP_t_bus0;next_BP_t_bus1 <= BP_t_bus1;next_BP_t_bus2 <=
BP_t_bus2;
			next_BP_t_eof <= BP_t_eof;next_BP_t_sof <=
BP_t_sof;next_BP_t_src_rdy <=
			BP_t_src_rdy;next_bus_na0 <= bus_na0;next_bus_na1 <=
bus_na1;next_bus_na2 <=
			bus_na2;next_dest_tx00 <= dest_tx00;next_dest_tx01 <= dest_tx01;
			next_dest_tx02 <= dest_tx02;next_dest_tx03 <= dest_tx03;next_dest_tx04
<=
			dest_tx04;next_dest_tx05 <= dest_tx05;next_dest_tx06 <= dest_tx06;
			next_dest_tx07 <= dest_tx07;next_dest_tx10 <= dest_tx10;next_dest_tx11
<=
			dest_tx11;next_dest_tx12 <= dest_tx12;next_dest_tx13 <= dest_tx13;
			next_dest_tx14 <= dest_tx14;next_dest_tx15 <= dest_tx15;next_dest_tx16
<=
			dest_tx16;next_dest_tx17 <= dest_tx17;next_t_next_bus0 <= t_next_bus0;
			next_t_next_bus1 <= t_next_bus1;next_t_next_bus2 <= t_next_bus2;

		BP_f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6, BP_f_add5,
BP_f_add4
			, BP_f_add3, BP_f_add2, BP_f_add1, BP_f_add0)));
		BP_f_cs <= (( std_logic_vector'(BP_f_cs2, BP_f_cs1, BP_f_cs0)));
		BP_t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6, BP_t_add5, BP_t_add4
			, BP_t_add3, BP_t_add2, BP_t_add1, BP_t_add0)));
		BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1, BP_t_bus0)));
		bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
		dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06, dest_tx05, dest_tx04
			, dest_tx03, dest_tx02, dest_tx01, dest_tx00)));
		dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16, dest_tx15, dest_tx14
			, dest_tx13, dest_tx12, dest_tx11, dest_tx10)));
```

```
t_next_bus <= (( std_logic_vector'(t_next_bus2, t_next_bus1, t_next_bus0)))
        ;


next_sreg<=END_T;
next_sreg1<=F_IDLE;

CASE sreg IS
    WHEN END_T =>
        next_sreg<=T_IDLE;
        next_BP_t_sof<='1';
        next_BP_t_eof<='1';
        next_BP_no_t<='1';
        next_BP_t_src_rdy<='1';

        bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
        t_next_bus <= (( std_logic_vector'(t_next_bus2, t_next_bus1,
t_next_bus0)
            ));
        BP_t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6,
BP_t_add5,
            BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)) +std_logic_vector'(
            "00000001") );
        BP_t_bus <= (std_logic_vector'("000"));
    WHEN START_T =>
        next_sreg<=T_ON;
        next_BP_t_sof<='0';
        next_BP_t_src_rdy<='0';
        bus_na <= (( std_logic_vector'(bus_na2, bus_na1, bus_na0)));
        BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1,
BP_t_bus0)));
        t_next_bus <= (( std_logic_vector'(t_next_bus2, t_next_bus1,
t_next_bus0)
            ));
        IF (( BP_no_t='1' )) THEN next_BP_no_t<='1';
        ELSE next_BP_no_t<='0';
        END IF;

        IF (( BP_t_eof='1' )) THEN next_BP_t_eof<='1';
        ELSE next_BP_t_eof<='0';
        END IF;

        BP_t_add <= ( ( (( std_logic_vector'(BP_t_add7, BP_t_add6,
BP_t_add5,
```

```
                                    BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)) +std_logic_vector'(
                                    "00000001") ) AND (  std_logic_vector'( t_add_en,
t_add_en, t_add_en,
                                    t_add_en, t_add_en, t_add_en, t_add_en, t_add_en)) ) OR
(( std_logic_vector'
                                    (BP_t_add7, BP_t_add6, BP_t_add5, BP_t_add4,
BP_t_add3, BP_t_add2, BP_t_add1,
                                     BP_t_add0)) AND (  std_logic_vector'( NOT t_add_en,
NOT t_add_en, NOT
                                    t_add_en, NOT t_add_en, NOT t_add_en, NOT t_add_en,
NOT t_add_en, NOT
                                    t_add_en)) ));
                        WHEN T_IDLE =>
                                IF ( t_new_bus0='1' ) OR ( t_new_bus1='1' ) OR ( t_new_bus2='1'
) THEN
                                    next_sreg<=T_SETUP;
                                    next_BP_no_t<='0';

                                    BP_t_add <= (( std_logic_vector'("01000000")));
                                    BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1,
BP_t_bus0)));
                                    IF (( BP_t_eof='1' )) THEN next_BP_t_eof<='1';
                                    ELSE next_BP_t_eof<='0';
                                    END IF;

                                    IF (( BP_t_sof='1' )) THEN next_BP_t_sof<='1';
                                    ELSE next_BP_t_sof<='0';
                                    END IF;

                                    IF (( BP_t_src_rdy='1' )) THEN next_BP_t_src_rdy<='1';
                                    ELSE next_BP_t_src_rdy<='0';
                                    END IF;

                                    t_next_bus <= (( std_logic_vector'(t_new_bus2,
t_new_bus1, t_new_bus0)))
                                            ;
                                    bus_na <= (( std_logic_vector'(t_new_bus2, t_new_bus1,
t_new_bus0)) AND
                                            (  std_logic_vector'(
bus_status0,bus_status1,bus_status2)));
                                ELSE
                                    next_sreg<=T_IDLE;
                                    next_BP_t_sof<='1';
                                    next_BP_t_eof<='1';
                                    next_BP_no_t<='1';
```

```
                                    next_BP_t_src_rdy<='1';

                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    t_next_bus <= (( std_logic_vector'(t_next_bus2,
t_next_bus1, t_next_bus0
                                        )));
                                    BP_t_add <= (std_logic_vector'("01000000"));
                                    BP_t_bus <= (std_logic_vector'("000"));
                                END IF;
                            WHEN T_ON =>
                                IF ( BP_t_add0='0' AND BP_t_add1='1' AND BP_t_add2='1'
AND BP_t_add3='1'
                                    AND BP_t_add4='1' AND BP_t_add5='0' AND
BP_t_add6='1' AND BP_t_add7='0' )
                                    THEN
                                    next_sreg<=END_T;
                                    next_BP_t_eof<='0';

                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    BP_t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6,
BP_t_add5,
                                        BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)) +
                                        (std_logic_vector'("00000001")));
                                    BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1,
BP_t_bus0)));
                                    t_next_bus <= (( std_logic_vector'(t_next_bus2,
t_next_bus1, t_next_bus0
                                        )));
                                    IF (( BP_no_t='1' )) THEN next_BP_no_t<='1';
                                    ELSE next_BP_no_t<='0';
                                    END IF;

                                    IF (( BP_t_sof='1' )) THEN next_BP_t_sof<='1';
                                    ELSE next_BP_t_sof<='0';
                                    END IF;

                                    IF (( BP_t_src_rdy='1' )) THEN next_BP_t_src_rdy<='1';
                                    ELSE next_BP_t_src_rdy<='0';
                                    END IF;

                                ELSE
                                    next_sreg<=T_ON;
                                    next_BP_t_sof<='1';
```

```
                                    bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                    BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1,
BP_t_bus0)));
                                    t_next_bus <= (( std_logic_vector'(t_next_bus2,
t_next_bus1, t_next_bus0
                                        )));
                                    IF (( BP_no_t='1' )) THEN next_BP_no_t<='1';
                                    ELSE next_BP_no_t<='0';
                                    END IF;

                                    IF (( BP_t_eof='1' )) THEN next_BP_t_eof<='1';
                                    ELSE next_BP_t_eof<='0';
                                    END IF;
                                    next_BP_t_src_rdy<='0';

                                    BP_t_add <= ( ( (( std_logic_vector'(BP_t_add7,
BP_t_add6, BP_t_add5,
                                        BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)) +std_logic_vector'(
                                        "00000001") ) AND (  std_logic_vector'( t_add_en,
t_add_en, t_add_en,
                                        t_add_en, t_add_en, t_add_en, t_add_en, t_add_en))
) OR  (( std_logic_vector'
                                        (BP_t_add7, BP_t_add6, BP_t_add5, BP_t_add4,
BP_t_add3, BP_t_add2, BP_t_add1,
                                         BP_t_add0)) AND (  std_logic_vector'( NOT
t_add_en, NOT t_add_en, NOT
                                        t_add_en, NOT t_add_en, NOT t_add_en, NOT
t_add_en, NOT t_add_en, NOT
                                        t_add_en)) ));
                            END IF;
                        WHEN T_SETUP =>
                            IF ( bus_na0='0' AND bus_na1='0' AND bus_na2='0' ) THEN
                                next_sreg<=START_T;
                                next_BP_t_sof<='1';
                                next_BP_t_src_rdy<='1';

                                bus_na <= (( std_logic_vector'(bus_na2, bus_na1,
bus_na0)));
                                BP_t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6,
BP_t_add5,
                                        BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)));
```

```
                                    t_next_bus <= (( std_logic_vector'(t_next_bus2,
t_next_bus1, t_next_bus0
                                        )));
                                    IF (( BP_no_t='1' )) THEN next_BP_no_t<='1';
                                    ELSE next_BP_no_t<='0';
                                    END IF;

                                    IF (( BP_t_eof='1' )) THEN next_BP_t_eof<='1';
                                    ELSE next_BP_t_eof<='0';
                                    END IF;

                                    BP_t_bus <= (( std_logic_vector'(t_next_bus2,
t_next_bus1, t_next_bus0))
                                        );
                                ELSE
                                    next_sreg<=T_SETUP;
                                    next_BP_no_t<='0';

                                    BP_t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6,
BP_t_add5,
                                        BP_t_add4, BP_t_add3, BP_t_add2, BP_t_add1,
BP_t_add0)));
                                    BP_t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1,
BP_t_bus0)));
                                    IF (( BP_t_eof='1' )) THEN next_BP_t_eof<='1';
                                    ELSE next_BP_t_eof<='0';
                                    END IF;

                                    IF (( BP_t_sof='1' )) THEN next_BP_t_sof<='1';
                                    ELSE next_BP_t_sof<='0';
                                    END IF;

                                    IF (( BP_t_src_rdy='1' )) THEN next_BP_t_src_rdy<='1';
                                    ELSE next_BP_t_src_rdy<='0';
                                    END IF;

                                    t_next_bus <= (( std_logic_vector'(t_new_bus2,
t_new_bus1, t_new_bus0)))
                                        ;
                                    bus_na <= (( std_logic_vector'(t_new_bus2, t_new_bus1,
t_new_bus0)) AND
                                        (  std_logic_vector'(
bus_status0,bus_status1,bus_status2)));
                                END IF;
                            WHEN OTHERS =>
                        END CASE;
```

```
                CASE sreg1 IS
                        WHEN f_analyze =>
                                next_sreg1<=F_ON;

                                dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06, dest_tx05,
                                        dest_tx04, dest_tx03, dest_tx02, dest_tx01, dest_tx00)));
                                dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16, dest_tx15,
                                        dest_tx14, dest_tx13, dest_tx12, dest_tx11, dest_tx10)));
                                BP_f_cs <= (( std_logic_vector'(BP_f_cs2, BP_f_cs1,
BP_f_cs0)));
                                BP_f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6,
BP_f_add5,
                                        BP_f_add4, BP_f_add3, BP_f_add2, BP_f_add1,
BP_f_add0)) +std_logic_vector'(
                                        "00000001"));
                                --My modifications
                                IF (bus0_busy='0') THEN BP_f_cs<=(std_logic_vector'("001"));
                                ELSIF (bus1_busy='0') THEN
BP_f_cs<=(std_logic_vector'("010"));
                                ELSIF (bus2_busy='0') THEN
BP_f_cs<=(std_logic_vector'("100"));
                                ELSE BP_f_cs<=(std_logic_vector'("000"));
                                END IF;
                                IF ((dest_tx00 = '1') OR (dest_tx01 = '1') OR (dest_tx02 = '1') OR
(dest_tx03 = '1')
                                        OR (dest_tx04 = '1') OR (dest_tx05 = '1') OR (dest_tx06 =
'1') OR (dest_tx07 = '1')) THEN
                                        BP_f_add<=(( std_logic_vector'("00000000")));
                                ELSIF ((dest_tx10 = '1') OR (dest_tx11 = '1') OR (dest_tx12 = '1')
OR (dest_tx13 = '1')
                                        OR (dest_tx14 = '1') OR (dest_tx15 = '1') OR (dest_tx16 =
'1') OR (dest_tx17 = '1')) THEN
                                        BP_f_add<=(( std_logic_vector'("00100000")));
                                ELSE BP_f_add<=(( std_logic_vector'("01000000")));
                                END IF;
                                ------------------
                        WHEN F_IDLE =>
                                IF ( rx0_sof='0' ) THEN
                                        next_sreg1<=f_analyze;

                                        BP_f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6,
BP_f_add5,
                                                BP_f_add4, BP_f_add3, BP_f_add2, BP_f_add1,
BP_f_add0)));
```

```
                                        dest_tx1 <= (( std_logic_vector'(f_dat7, f_dat6, f_dat5,
f_dat4, f_dat3,
                                                f_dat2, f_dat1, f_dat0)) AND (
std_logic_vector'(BP_tx1_mask7, BP_tx1_mask6,
                                                BP_tx1_mask5, BP_tx1_mask4, BP_tx1_mask3,
BP_tx1_mask2, BP_tx1_mask1,
                                               BP_tx1_mask0)));
                                        dest_tx0 <= (( std_logic_vector'(f_dat7, f_dat6, f_dat5,
f_dat4, f_dat3,
                                                f_dat2, f_dat1, f_dat0)) AND (
std_logic_vector'(BP_tx0_mask7, BP_tx0_mask6,
                                                BP_tx0_mask5, BP_tx0_mask4, BP_tx0_mask3,
BP_tx0_mask2, BP_tx0_mask1,
                                               BP_tx0_mask0)));
                                ELSE
                                        next_sreg1<=F_IDLE;

                                        dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                                dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                        dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                                dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                        BP_f_cs <= (( std_logic_vector'("000")));
                                        BP_f_add <= (std_logic_vector'("00000000"));
                                END IF;
                        WHEN F_ON =>
                                IF ( BP_f_add0='1' AND BP_f_add1='1' AND BP_f_add2='1'
AND BP_f_add3='1'
                                        AND BP_f_add4='1' AND BP_f_add6='0' AND
BP_f_add7='0' ) THEN
                                        next_sreg1<=F_IDLE;

                                        dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                                dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                        dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                                dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                        BP_f_cs <= (( std_logic_vector'(BP_f_cs2, BP_f_cs1,
BP_f_cs0)));
```

```
                                        BP_f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6,
BP_f_add5,
                                                BP_f_add4, BP_f_add3, BP_f_add2, BP_f_add1,
BP_f_add0)) +std_logic_vector'(
                                                "00000001"));
                                ELSE
                                        next_sreg1<=F_ON;

                                        dest_tx0 <= (( std_logic_vector'(dest_tx07, dest_tx06,
dest_tx05,
                                                dest_tx04, dest_tx03, dest_tx02, dest_tx01,
dest_tx00)));
                                        dest_tx1 <= (( std_logic_vector'(dest_tx17, dest_tx16,
dest_tx15,
                                                dest_tx14, dest_tx13, dest_tx12, dest_tx11,
dest_tx10)));
                                        BP_f_cs <= (( std_logic_vector'(BP_f_cs2, BP_f_cs1,
BP_f_cs0)));
                                        BP_f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6,
BP_f_add5,
                                                BP_f_add4, BP_f_add3, BP_f_add2, BP_f_add1,
BP_f_add0)) +std_logic_vector'(
                                                "00000001"));
                                END IF;
                        WHEN OTHERS =>
                END CASE;

                next_BP_f_add7 <= BP_f_add(7);
                next_BP_f_add6 <= BP_f_add(6);
                next_BP_f_add5 <= BP_f_add(5);
                next_BP_f_add4 <= BP_f_add(4);
                next_BP_f_add3 <= BP_f_add(3);
                next_BP_f_add2 <= BP_f_add(2);
                next_BP_f_add1 <= BP_f_add(1);
                next_BP_f_add0 <= BP_f_add(0);
                next_BP_f_cs2 <= BP_f_cs(2);
                next_BP_f_cs1 <= BP_f_cs(1);
                next_BP_f_cs0 <= BP_f_cs(0);
                next_BP_t_add7 <= BP_t_add(7);
                next_BP_t_add6 <= BP_t_add(6);
                next_BP_t_add5 <= BP_t_add(5);
                next_BP_t_add4 <= BP_t_add(4);
                next_BP_t_add3 <= BP_t_add(3);
                next_BP_t_add2 <= BP_t_add(2);
                next_BP_t_add1 <= BP_t_add(1);
                next_BP_t_add0 <= BP_t_add(0);
```

```
        next_BP_t_bus2 <= BP_t_bus(2);
        next_BP_t_bus1 <= BP_t_bus(1);
        next_BP_t_bus0 <= BP_t_bus(0);
        next_bus_na2 <= bus_na(2);
        next_bus_na1 <= bus_na(1);
        next_bus_na0 <= bus_na(0);
        next_dest_tx07 <= dest_tx0(7);
        next_dest_tx06 <= dest_tx0(6);
        next_dest_tx05 <= dest_tx0(5);
        next_dest_tx04 <= dest_tx0(4);
        next_dest_tx03 <= dest_tx0(3);
        next_dest_tx02 <= dest_tx0(2);
        next_dest_tx01 <= dest_tx0(1);
        next_dest_tx00 <= dest_tx0(0);
        next_dest_tx17 <= dest_tx1(7);
        next_dest_tx16 <= dest_tx1(6);
        next_dest_tx15 <= dest_tx1(5);
        next_dest_tx14 <= dest_tx1(4);
        next_dest_tx13 <= dest_tx1(3);
        next_dest_tx12 <= dest_tx1(2);
        next_dest_tx11 <= dest_tx1(1);
        next_dest_tx10 <= dest_tx1(0);
        next_t_next_bus2 <= t_next_bus(2);
        next_t_next_bus1 <= t_next_bus(1);
        next_t_next_bus0 <= t_next_bus(0);
END PROCESS;

PROCESS (BP_no_t)
BEGIN
        IF (( BP_no_t='1' )) THEN no_t<='1';
        ELSE no_t<='0';
        END IF;
END PROCESS;

PROCESS (BP_t_eof)
BEGIN
        IF (( BP_t_eof='1' )) THEN t_eof<='1';
        ELSE t_eof<='0';
        END IF;
END PROCESS;

PROCESS (BP_t_sof)
BEGIN
        IF (( BP_t_sof='1' )) THEN t_sof<='1';
        ELSE t_sof<='0';
        END IF;
```

```
END PROCESS;

PROCESS (BP_t_src_rdy)
BEGIN
	IF (( BP_t_src_rdy='1' )) THEN t_src_rdy<='1';
	ELSE t_src_rdy<='0';
	END IF;
END PROCESS;

PROCESS (BP_tx0_mask)
BEGIN
	BP_tx0_mask <= (std_logic_vector'("00001110"));
	BP_tx0_mask0 <= BP_tx0_mask(0);
	BP_tx0_mask1 <= BP_tx0_mask(1);
	BP_tx0_mask2 <= BP_tx0_mask(2);
	BP_tx0_mask3 <= BP_tx0_mask(3);
	BP_tx0_mask4 <= BP_tx0_mask(4);
	BP_tx0_mask5 <= BP_tx0_mask(5);
	BP_tx0_mask6 <= BP_tx0_mask(6);
	BP_tx0_mask7 <= BP_tx0_mask(7);
END PROCESS;

PROCESS (BP_tx1_mask)
BEGIN
	BP_tx1_mask <= (std_logic_vector'("11110000"));
	BP_tx1_mask0 <= BP_tx1_mask(0);
	BP_tx1_mask1 <= BP_tx1_mask(1);
	BP_tx1_mask2 <= BP_tx1_mask(2);
	BP_tx1_mask3 <= BP_tx1_mask(3);
	BP_tx1_mask4 <= BP_tx1_mask(4);
	BP_tx1_mask5 <= BP_tx1_mask(5);
	BP_tx1_mask6 <= BP_tx1_mask(6);
	BP_tx1_mask7 <= BP_tx1_mask(7);
END PROCESS;

PROCESS (BP_f_add0,BP_f_add1,BP_f_add2,BP_f_add3,BP_f_add4,BP_f_add5,
	BP_f_add6,BP_f_add7,f_add)
BEGIN
	f_add <= (( std_logic_vector'(BP_f_add7, BP_f_add6, BP_f_add5, BP_f_add4,
		BP_f_add3, BP_f_add2, BP_f_add1, BP_f_add0)));
	f_add0 <= f_add(0);
	f_add1 <= f_add(1);
	f_add2 <= f_add(2);
	f_add3 <= f_add(3);
	f_add4 <= f_add(4);
	f_add5 <= f_add(5);
```

```
		f_add6 <= f_add(6);
		f_add7 <= f_add(7);
	END PROCESS;

	PROCESS (BP_f_cs0,BP_f_cs1,BP_f_cs2,f_cs)
	BEGIN
		f_cs <= (( std_logic_vector'(BP_f_cs2, BP_f_cs1, BP_f_cs0)));
		f_cs0 <= f_cs(0);
		f_cs1 <= f_cs(1);
		f_cs2 <= f_cs(2);
	END PROCESS;

	PROCESS (BP_t_add0,BP_t_add1,BP_t_add2,BP_t_add3,BP_t_add4,BP_t_add5,
		BP_t_add6,BP_t_add7,t_add)
	BEGIN
		t_add <= (( std_logic_vector'(BP_t_add7, BP_t_add6, BP_t_add5, BP_t_add4,
			BP_t_add3, BP_t_add2, BP_t_add1, BP_t_add0)));
		t_add0 <= t_add(0);
		t_add1 <= t_add(1);
		t_add2 <= t_add(2);
		t_add3 <= t_add(3);
		t_add4 <= t_add(4);
		t_add5 <= t_add(5);
		t_add6 <= t_add(6);
		t_add7 <= t_add(7);
	END PROCESS;

	PROCESS (BP_t_bus0,BP_t_bus1,BP_t_bus2,t_bus)
	BEGIN
		t_bus <= (( std_logic_vector'(BP_t_bus2, BP_t_bus1, BP_t_bus0)));
		t_bus0 <= t_bus(0);
		t_bus1 <= t_bus(1);
		t_bus2 <= t_bus(2);
	END PROCESS;
END BEHAVIOR;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY INTF IS
	PORT (bus_status : IN std_logic_vector (2 DOWNTO 0);
		f_add : OUT std_logic_vector (7 DOWNTO 0);
		f_cs : OUT std_logic_vector (2 DOWNTO 0);
```

```
                f_dat : IN std_logic_vector (7 DOWNTO 0);
                t_add : OUT std_logic_vector (7 DOWNTO 0);
                t_bus : OUT std_logic_vector (2 DOWNTO 0);
                t_dat : IN std_logic_vector (7 DOWNTO 0);
                t_new_bus : IN std_logic_vector (2 DOWNTO 0);
                CLK,RESET,rx0_sof,t_add_en,bus0_busy,bus1_busy,bus2_busy: IN std_logic;
                no_t,t_eof,t_sof,t_src_rdy : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF INTF IS
        COMPONENT SHELL_INTF
                PORT
(CLK,bus_status0,bus_status1,bus_status2,f_dat0,f_dat1,f_dat2,f_dat3,f_dat4,f_dat5,f_dat6,
                        f_dat7,RESET,rx0_sof,t_add_en,t_dat0,t_dat1,t_dat2,t_dat3,t_dat4,t_dat5,
                        t_dat6,t_dat7,t_new_bus0,t_new_bus1,t_new_bus2,bus0_busy,bus1_busy,
                        bus2_busy: IN std_logic;

        f_add0,f_add1,f_add2,f_add3,f_add4,f_add5,f_add6,f_add7,f_cs0,f_cs1,f_cs2,

        no_t,t_add0,t_add1,t_add2,t_add3,t_add4,t_add5,t_add6,t_add7,t_bus0,t_bus1,
                                t_bus2,t_eof,t_sof,t_src_rdy : OUT std_logic);
        END COMPONENT;
BEGIN
        SHELL1_INTF : SHELL_INTF PORT MAP
(CLK=>CLK,bus_status0=>bus_status(0),bus_status1
                =>bus_status(1),bus_status2=>bus_status(2),f_dat0=>
                f_dat(0),f_dat1=>f_dat(1),f_dat2=>f_dat(2),f_dat3=>f_dat(3),f_dat4=>f_dat(4),

        f_dat5=>f_dat(5),f_dat6=>f_dat(6),f_dat7=>f_dat(7),RESET=>RESET,rx0_sof=>
                rx0_sof,t_add_en=>t_add_en,t_dat0=>t_dat(0),t_dat1=>t_dat(1),t_dat2=>t_dat(2)
                ,t_dat3=>t_dat(3),t_dat4=>t_dat(4),t_dat5=>t_dat(5),t_dat6=>t_dat(6),t_dat7=>
                t_dat(7),t_new_bus0=>t_new_bus(0),t_new_bus1=>t_new_bus(1),t_new_bus2=>

        t_new_bus(2),f_add0=>f_add(0),f_add1=>f_add(1),f_add2=>f_add(2),f_add3=>f_add

        (3),f_add4=>f_add(4),f_add5=>f_add(5),f_add6=>f_add(6),f_add7=>f_add(7),f_cs0

        =>f_cs(0),f_cs1=>f_cs(1),f_cs2=>f_cs(2),no_t=>no_t,t_add0=>t_add(0),t_add1=>

        t_add(1),t_add2=>t_add(2),t_add3=>t_add(3),t_add4=>t_add(4),t_add5=>t_add(5),

        t_add6=>t_add(6),t_add7=>t_add(7),t_bus0=>t_bus(0),t_bus1=>t_bus(1),t_bus2=>
                t_bus(2),t_eof=>t_eof,t_sof=>t_sof,t_src_rdy=>t_src_rdy,
                bus0_busy=>bus0_busy,bus1_busy=>bus1_busy,bus2_busy=>bus2_busy);
END BEHAVIOR;
```