INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

VHDL DESIGN AND IMPLEMENTATION OF A LAN ON A CHIP

By

Xiao Gang Deng

A Thesis Submitted to
The school of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science
in Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

Acknowledgment

I would like to express my gratitude and appreciation to my supervisors Dr. Tet Yeap and Dr. A. Karmouch for their detailed guidance, advises, encouragement and continuous support through out the entire thesis project.

I would also like to thank lab technician and system administrator for their cheerful help at various stages in the project.

Abstract

With ever increasing usage of Internet and more demand for real-time applications, the intrinsic shortcoming of Ethernet becomes more and more apparent. Ethernet architecture combined with the Medium Access Control protocol can not guarantee a maximum time delay variation, jitter, which is critical to real-time applications. In this thesis, a novel LAN architecture is proposed, designed and implemented with field programmable gate array (FPGA) device as target. This LAN on a chip design is simple, scaleable and modular. The preliminary simulation shows that it reduces the impact of packet collision on the LAN and improves the performance in terms of reducing and limiting jitter on real-time traffic.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Over the last few decades, LAN (Local Area Network) has been a widely accepted form of data communication network in the world. In general, there are three LAN topologies, namely, bus, ring and star. Ethernet and Token Ring are examples of bus and ring topologies, respectively. Because it is simple and easy to configure, Ethernet has become the most popular LAN topology. Ethernet uses a shared bus as data transmission medium and the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) is adopted as the MAC (Medium Access Control) protocol for accessing this common bus. A large number of applications run on Ethernet, such as ftp, email and world wide web.

1.1 Thesis Motivation

However, real-time applications, such as VOD (Video On Demand) and electronic shopping, are very sensitive to time delay variation, so called jitter, that occurs during data transmission. Being a shared data bus, Ethernet can not guarantee a uniform delay or maximum delay for any data stream making it unsuitable for such applications. Real-time

applications pose a very strict time delay limit on any type data networks. Increasing the bit rate of Ethernet eases the problem of controlling the time delay to some degree. However, the shared data bus architecture together with CSMA/CD MAC protocol make it impossible for Ethernet to overcome this problem completely. Many alternative LAN architectures are being studied in order to improve or replace the current Ethernet architecture.

1.2 Thesis Objective

The objective of this thesis is to introduce a new scalable and simple LAN architecture that guarantees a maximum time delay on data transmission and to implement the design on an Altera Flex10k FPGA (Field Programmable Gate Array) device.

1.3 Thesis Organization

A brief literature overview on LAN architecture and Ethernet is presented in chapter 2. The FERMTOR architecture is also presented in chapter 2. In chapter 3, the overall structure of the proposed LAN architecture is discussed. The circuit level design of the alternative architecture is presented, in details, in chapter 4. The analysis of Ethernet architecture by computer simulation is presented in chapter 5. VHDL simulation of the LAN on a chip architecture is presented in chapter 6. Conclusions and future work are discussed in chapter 7.

1.4 Main Contribution

The main contributions of this thesis are:

- Design of a simple, modular and scalable architecture for LAN that guarantees the maximum time delay in real-time application.
- Implementation of the new architecture on Altera FLEX10K FPGA device.
- Simulation the implementation in ModelTech※s VHDL simulation environment.

Chapter 2

Background and Literature Review

2.1 Description of LAN

With the development in computer and data communications industry, LAN (Local Area Network) becomes a critical part of data networks. A local area network is a data communications network (or computer network) that is confined to a limited geographical boundary, such as a room, a building or a campus. The typical use of LAN is to connect various DTEs (data terminal equipment) to facilitate the communications between these devices and the sharing of resources on servers, such as file server, or peripheral devices, such as printer.

There are several types of physical medium being used to construct a LAN. Twisted pair of copper wires is the most widely used due to its low cost and easy installation and maintenance. Coaxial cable is also widely used. Coaxial cable consists of a center wire surrounded by insulation and then a ground shield of braided wire. The shield minimizes the electrical and radio frequency interference. With the increasing demand on bandwidth

of LAN traffic, fiber optic cables also become a choice of physical medium of LAN. Fiber optic cables have much greater bandwidth than metal cables, yet fiber optic cables are much less susceptible than metal cables to interference. Fiber optic cable is lighter and thinner, therefore easy to install and maintain. One advantage of fiber optic cables is the data can be transmitted digitally rather than in analog signal.

Several LAN protocols including Ethernet, Fast Ethernet, switched Ethernet, Token Ring, FDDI (Fiber Distributed Data Interface) are in existence. Ethernet is the most commonly implemented LAN standard and it supports data transfer rate of 10 Mbps (megabits per second). Fast Ethernet and FDDI can support bit rate of 100 Mbps.

Typical applications that runs on LAN are of client/server type, such as ftp, email, Internet, World Wide Web.

2.2 Description of Ethernet

Ethernet is a LAN standard developed by Xerox Corporation in cooperation with DEC and Intel in 1976. Ethernet uses bus or star topology and support data transfer rate of 10 Mbps, in the case of 10Base-T standard. A newer version of the Ethernet, called Fast Ethernet (or 100Base-T) supports data transfer rate of 100 Mbps. The newest version of Ethernet, Gigabit Ethernet, supports data transfer rate of 1 Gbps. A diagram of Ethernet structure with bus topology is shown in Figure 2-1.

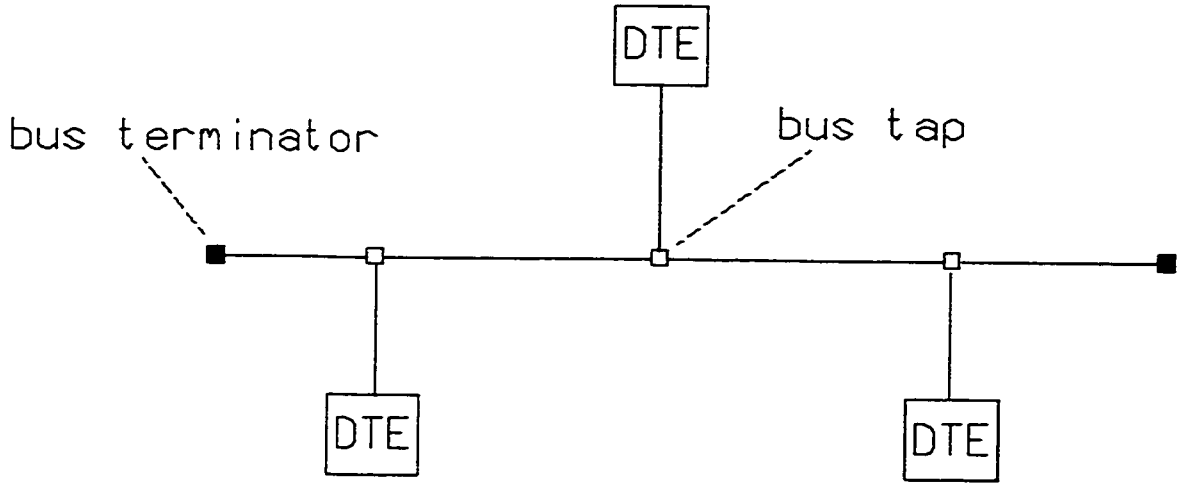


Figure 2-1 : Ethernet data bus

Ethernet uses CSMA/CD (Carrier Sense Multiple Access with Collision Detection) as it MAC (Medium Access Control) protocol to handle simultaneous access demands and contention problems.

2.3 Description of CSMA/CD

CSMA/CD is the most commonly used medium access control technique for bus and star topologies. The rules for CSMA/CD are as follows [1]:

1. If the medium is idle, transmit; otherwise, go to step 2;
2. If the medium is busy, continue to listen until the channel is idle, then transmit immediately.
3. If a collision is detected during transmission, transmit a brief jamming signal to assure that all stations know that there has been a collision and then cease transmission,
4. after transmitting the jamming signal, wait a random amount of time, then attempt to transmit again (repeat from step 1).

Here, CSMA/CD employs 1-persistent persistence algorithm. This persistence algorithm is used by Ethernet and is in the IEEE802 standard. The 1-persistent persistence algorithm says:

1. If the medium is idle, transmit; otherwise, go to step 2.
2. If the medium is busy, continue to listen until the channel is sensed idle; then transmit immediately.

2.4 Impact of Ethernet Protocol on Real-time Applications

From previous examination of CSMA/CD protocol, we can see that when two DTEs try to access the data bus at the same time, i.e. when there is collision, both DTEs cease transmission and wait for a random amount of time before trying to access the medium and transmit again. The duration of this period of holding time is not deterministic but a random value. That is to say, under contention situations, some packets may suffer long delay than others, and this delay variation is not under any control.

As we know, real-time applications, such as audio and video applications, have strict requirement on jitter, time delay variation. In real-time applications, a slight time delay is not the biggest concern to end users. After all, any data transmission introduces time delay. As long as every data packets arrive at the user end with approximately equal time delay, the perception of audio or video information is not affected. Jitter in data transmission results in garbled conversation or jerky video.

Jitter is a significant problem that Ethernet networks are facing, because CSMA/CD protocol gives no guarantee on time delay for data packet transmission. In Chapter 5, the effect of jitter in an Ethernet LAN will be demonstrated by computer simulations.

2.5 Description of Token Ring

Token Ring is another popular LAN topology. Token Ring specification was developed by IBM. Token Ring uses a ring type of bus to connect all the computers on the LAN. A

token, which is a special bit pattern, travels around the ring. All computers attached on the ring listen for the token. To send a message, a computer catches the token, attaches a message to it, and then, lets it continue the travel around the network. After a computer finishes the transmisstion, it passes token to the next computer on the ring. A diagram of Token Ring network is shown in Figure 2-2.

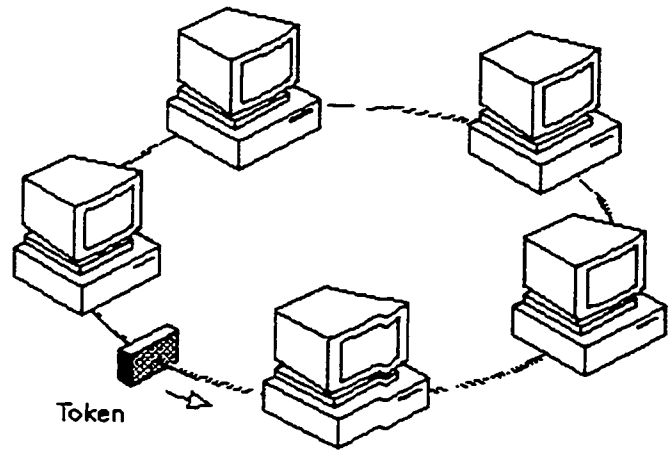


Figure 2-2 : Token Ring network

2.6 FERMTOR Architecture

The FERMTOR architecture was proposed by Dr. Wayne M. Lucks, in his Ph.D. thesis at University of Toronto, as a multiprocessor computer structure intended for applications in multiprogramming environment [2]. FERMTOR architecture is also a general purpose multiprocessor architecture that can be tuned to many applications. FERMTOR architecture has several features in common with a number of data-flow machine architectures [3][4][5]. It is a practical multiprocessor whose communications hardware is much less complex than that of a MIMD (Mulltiple Instruction stream, Multiple Data stream) crossbar or Banyan network. FERMTOR's ring and common-access bus structure resemble the topology of the EMMA architecture and Generalized Control Flow (GCF) machine [6][7].

2.6.1 Description of FERMTOR Architecture

Figure 2-3 illustrates the basic structure of FERMTOR architecture. It is a part ring, part shared common-access bus architecture [2][8][9][10]. The shared common-access bus is called P-Bus which is a ring-like pipelined bus structure. Processors are connected to the P-Bus. Parallel data flows synchronously between the latches along the bus. P-Bus is *n* bits wide, where *n* is the length of a packet being relayed on the bus. Each latch, bus segment between latches, and group of processors following it is known collectively as a station. Every processor on the P-Bus has a unique address by which it can be referenced. The packet is the basic unit of communication among all processors. A packet contains the following fields:

- ***source*** - the P-Bus address of the transmitting processor.
- ***destination*** - the P-Bus address of the receiving processor.
- ***operation*** - specification of the nature of the packet.
- ***operand*** - data to be operated on.

Three status bits are also appended to specify the presence or absence of a packet in the time slot and if the packet has been successfully received by the destination station or not.

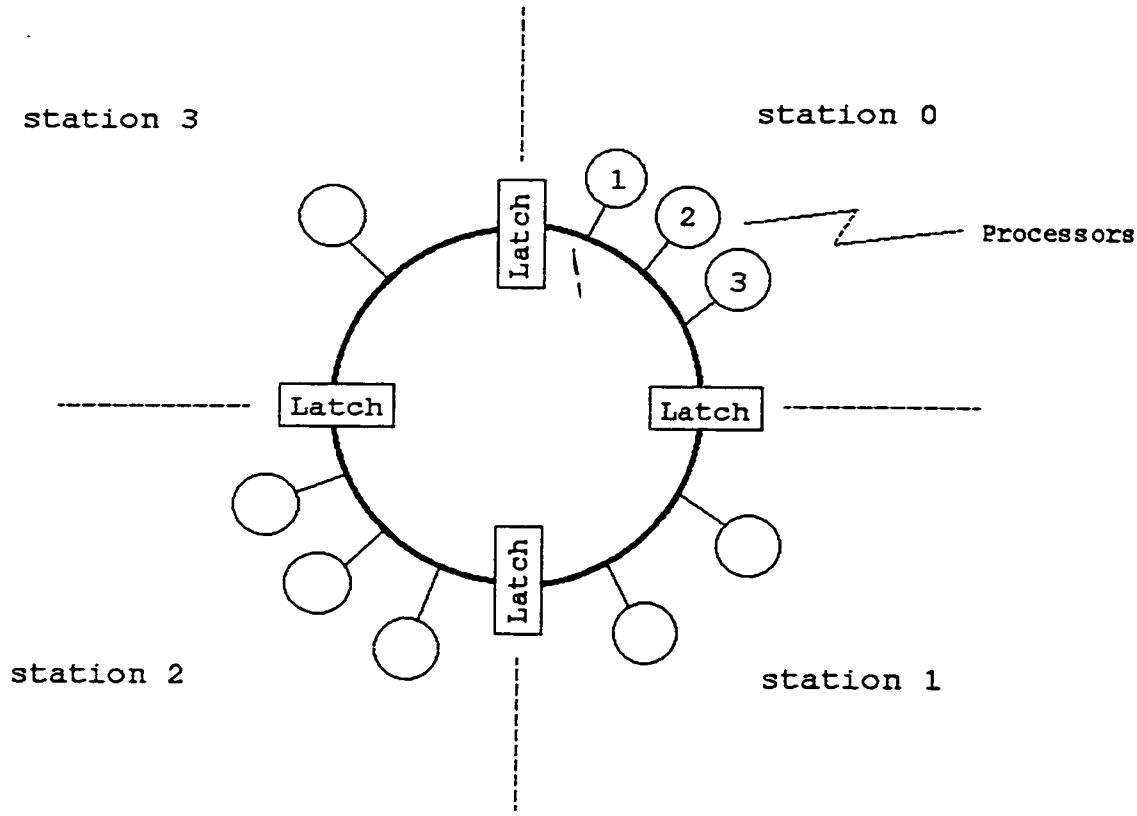


Figure 2-3 : FERMTOR architecture

2.6.2 P-Bus Operation

Figure 2-4 is a block diagram of one station and the P-Bus interface of two processors [2][8][9][10]. There is a station manager at each station. The station manager controls the flow of packets among local processors and between the neighboring stations. The station latches are synchronously clocked at a fixed bus clock cycle rate. Within each bus clock cycle, the station manager decides whether a transfer in progress can be processed further, or if a new transfer can be started.

When a processor wants to transmit a packet, it sends a Request signal to the station manager. The station manager arbitrates the processor requests and send an Enable signal to the selected processor as soon as the first empty packet arrives from the previous

station. The requested transfer can be either a local transfer between processors in the same station or a nonlocal transfer between processors at different stations.

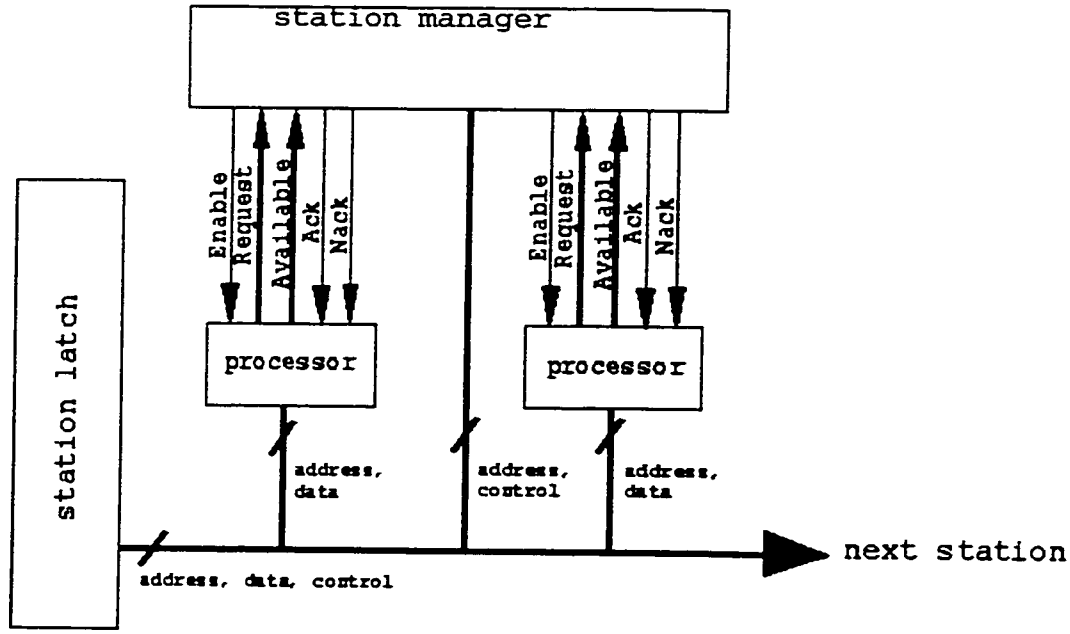


Figure 2-4 : station and its P-Bus interface

In the local transfer scheme, since the packet's destination is local, the station manager tries to transmit the packet to the destination processor as soon as possible when the available signal is received by the station manager from the destination processor. If the packet is accepted by the destination processor, an Ack (acknowledge) signal is sent to the transmitting processor by the station manager. If the packet is not accepted by the destination processor, i.e., the destination processor is not available for accepting any packets, a Nack (negative acknowledge) is sent to the transmitting processor. Not matter the transaction is successful or not, the packet slot occupied by this transaction is then marked Empty.

In the case of nonlocal transfer, the packet is transmitted to the next station along the ring. Each station checks destination address field of the incoming packet to see if it is the destination station. After the destination station is reached, the station manager tries to transmit the packet to the destination processor in the same way as described above. If the transmission is successful, the packet is marked Received. Otherwise, the packet is marked Not Received, using one of the appended flags. When the packet returns to the transmitting station, the station manager at the transmitting station sends to the transmitting processor either an Ack signal, if the returning packet was marked Received, or Nack signal, if the returning packet was marked Not Received.

For any unsuccessful transfers, the processor regenerates a Request signal until the transfer succeeds. The P-Bus protocol allows a processor to have only one packet active on the ring at one time, to prevent it from dominating ring traffic.

In this thesis, our interest is on the nonlocal transfer since we would like to build a LAN architecture to facilitate communications among stations not within a single station.

2.6.3 Description of Station Manager

The block diagram of a station manager is shown in Figure 2-5. The station manager is composed of four principal parts. The station latch holds the data packets transmitted among the stations during a bus clock cycle and latches the data packet from the preceding station when the next bus clock cycle comes. In case of multiple processors in a single station, the arbitration unit decides which packet is sent onto the bus. If the incoming ring packet is full, then the ring packet takes the priority. If the incoming ring packet is empty, then the arbitration units arbitrates processor requests for the bus. The receive data control unit transmits any packets destined for this station to the addressed processor by clocking the ring packet data into the processor's buffer. The transmit data control unit checks if the incoming ring packet is appended with Received or Not Received flag and informs the transmitting processor with an Ack or a Nack signal, respectively [2][8][9][10].

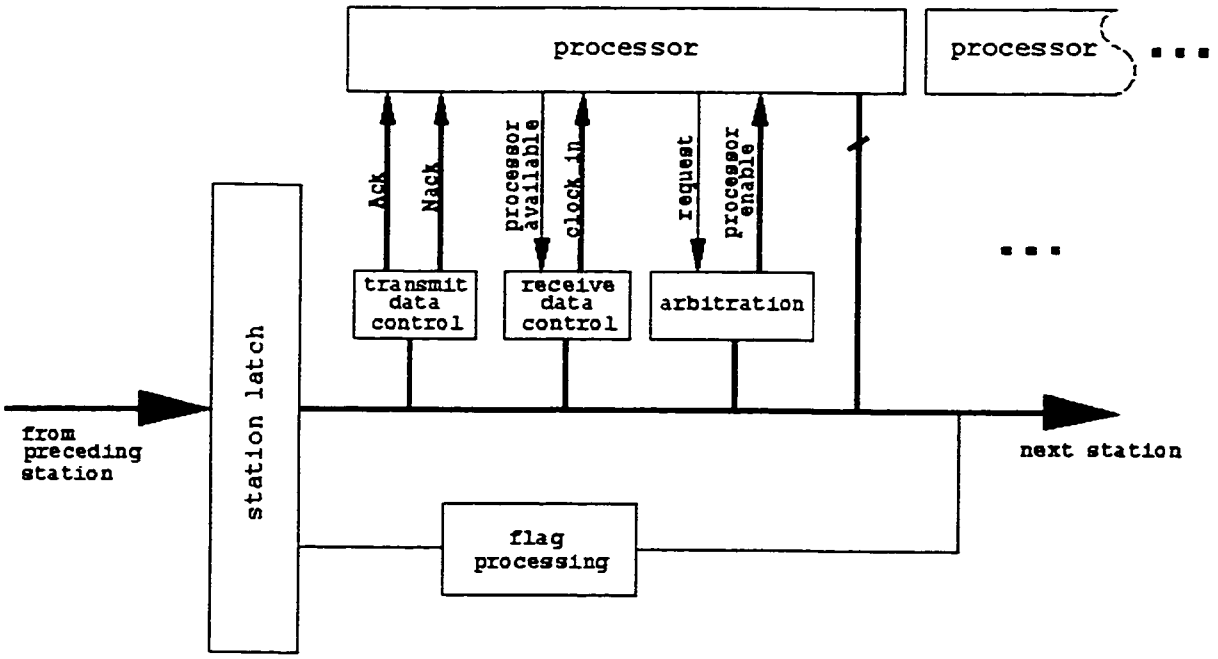


Figure 2-5 : station manager

2.6.4 Description of Processor

The block diagram of a processor is depicted in Figure 2-6. The P-Bus contains two parts. The input and output buffers that contain data being received and waiting to be transmitted. The control module that handles the following control signals [2][8][9][10]:

- ***Request*** : an output request to the station manager, indicating that there is data in the output buffer to be transmitted.
- ***Processor Available*** : an output-status flag indicating that the processor is able to receive another packet.
- ***Processor Enable*** : an input from the station manager that causes the processor to place a requested transmission onto the P-Bus.

- ***Clock In*** : an input from the station manager that sends a packet form the P-Bus to the processor.
- ***Acknowledge*** : an input from the station manager indicating that the last request was successful.
- ***Negative Acknowledge*** : an input from the station manager indicating that the last request was not successful.

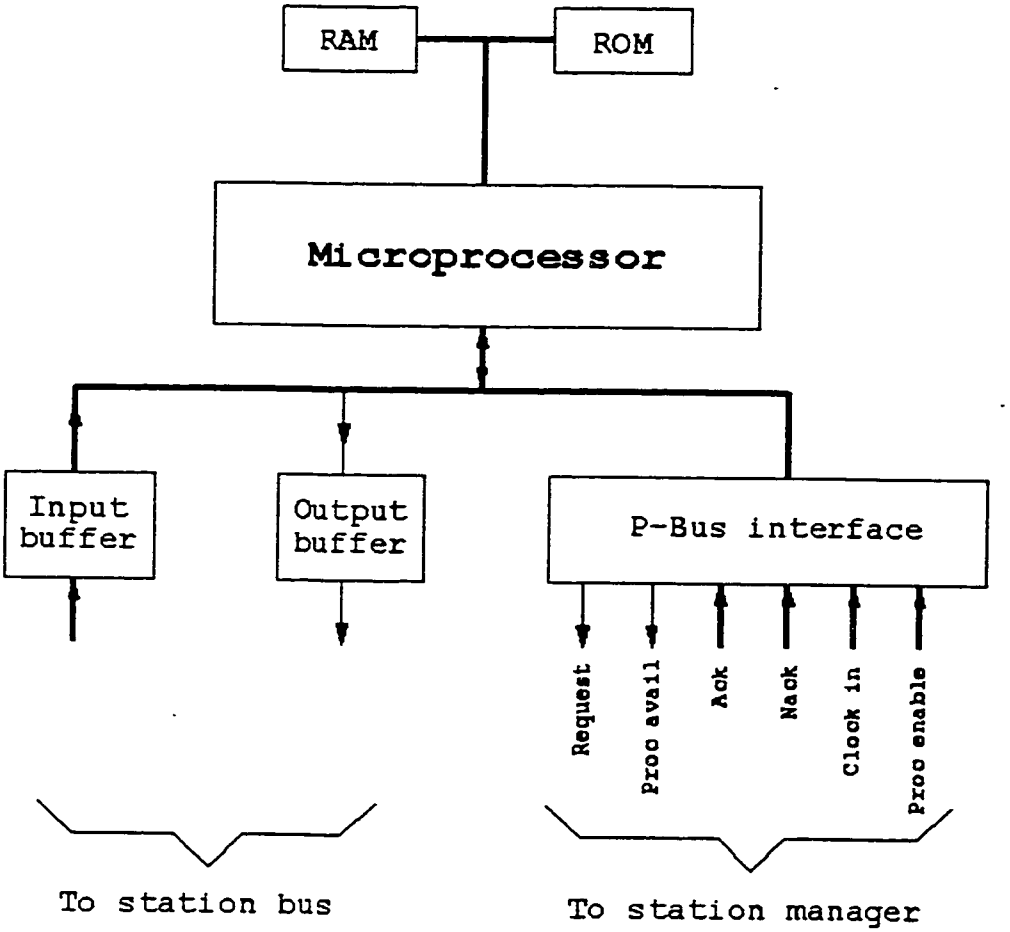


Figure 2-6 : structure of a processor

2.6.5 FERMTOR Architecture Characteristics

FERMTOR is a simple, flexible and easily extendible architecture. Each station on the bus is a self-contained unit. This means that additional stations can be easily added to the P-Bus. The hardware complexity of FERMTOR is directly proportional to the number of stations while there is not limitation on the actual size of the ring. However, as the number of stations increases, the time required for a packet to traverse the ring becomes larger.

Couple of interesting features of P-Bus operation are worth noticing. First, once the a packet returns to the transmitting station, it must leave that station marked as Empty. This ensures that one station will not hog the ring transfers. Second, at any time, each processor can be the source of no more then one transfer on the bus. This avoids hogging of the bus by a single active processor and also simplifies retransmission since no packet numbering scheme is required.

Another important feature of P-Bus operation is that multiple stations can access the ring bus at the same time. When more than one station access the bus, their data occupy different packet slot. Since the packet slots are isolated from each other, the collision between the data packets from different station do not happen. This feature gives us an opportunity to find a solution to the problem of collision, hence jitter, facing the Ethernet bus.

2.6.6 Conclusion

The FERMTOR is a simple flexible extendible architecture that can be tuned to become a LAN architecture. The P-Bus ring structure and its protocol make the simultaneous access of the bus without collision possible.

Chapter 3

LAN on A Chip

The problem of traditional Ethernet architecture is the time delay variation when collision of packets happens on the common bus., thus causing the architecture to be unsuitable for real-time applications, such as transmission of video. The FERMTOR architecture brings some promising features that can avoid collision on the ring bus and bring the time delay variation under control. In this thesis, the generic FERMTOR architecture is tuned to become a LAN architecture. The implementation of the new LAN design fits on a FPGA chip, resulting a LAN on a chip.

3.1 Overview of The Proposed LAN Architecture

A block diagram of the LAN on a chip structure is shown in Figure 3-1. Four transceiving stations are connected by a data bus to form a ring. The data bus is of the form of the P-Bus in the FERMTOR architecture. A DTE can be connected to a tranceiving station's I/O interface to have access to the LAN. The following signal ports appear on the I/O interface of a tranceiving station.

- ***data input***
- ***data input clock***
- ***data input ready***
- ***data output***
- ***data output clock***
- ***data output ready***

Ethernet interface adapter can be connected to the I/O interface to facilitate data transmission over a physical medium such as twisted pair cable. In this thesis, this interface was designed to support up to 10 Mbps bit rate, which is the bit rate supported by 10Base-T Ethernet.

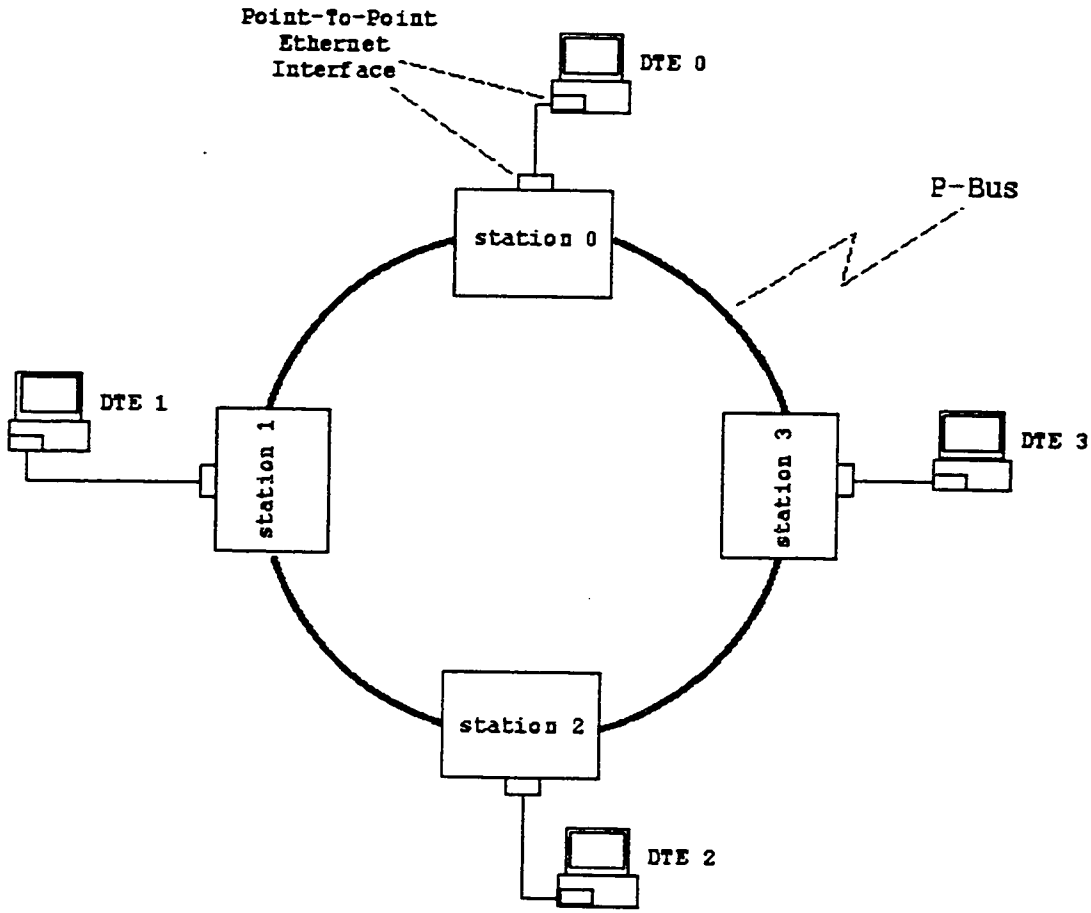


Figure 3-1 : LAN on a chip structure

The data bus is a parallel pipelined bus with a width of 33 bits. 33 bits is the standard size of a data packet that is transferred on the ring. The packet length could be any size. This 33 bits length is the result of a compromise between ring bus data rate and efficiency of each packet. On one hand, we want each packet to have low overhead, i.e. longer data field, short header. On the other hand, a longer packet requires longer ring bus clock cycle which slows things down. The size chosen here seems to be a good compromise to address both needs. Figure 3-2 shows the format of a data packet. A standard data packet contains the following fields.

- ***group indicator (1 bit)*** - this bit indicates to the receiver whether the current packet is the last segment of the entire Ethernet data packet that is supposed to be received by the receiving station.
- ***source station address (3 bits)*** - provides the binary address of a transmitting station on the ring.
- ***destination station address (3 bits)*** - provides the binary address of a receiving station on the ring.
- ***packet type indicator (2 bits)*** - indicates the type of the current data packet.

  "00" - empty packet

  "01" - Nack packet

  "10" - Ack packet

  "11" - real data packet
- ***data field (24 bits)*** - contains the 24 bits of data that is being transferred.

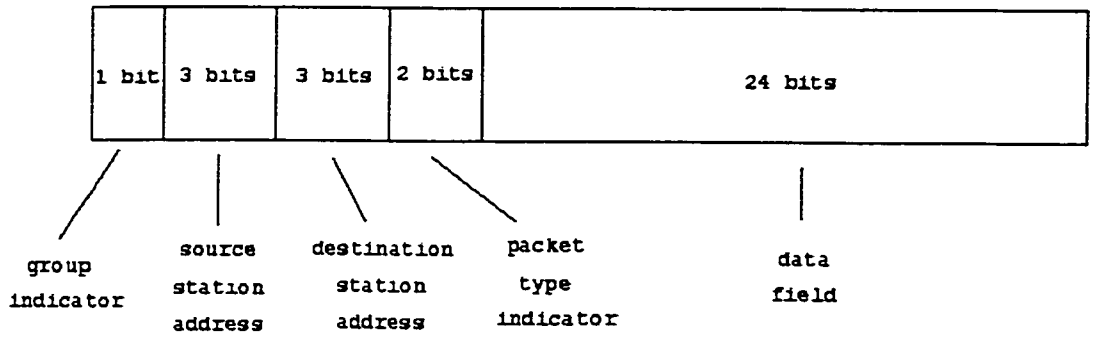


Figure 3-2 : data packet format

Four types of packet are used on the LAN. They are:

- ***real data packet*** - this type of packet carries real data that is transmitted in the data field.
- ***acknowledgment packet*** - this type of packet carries no real data but indicates to the transmitting station that the last transmission was successful.
- ***negative acknowledgment packet*** - this type of packet carries no real data but informs the transmitting station that the last transmission was not successful.
- ***empty packet*** - this type of packet has empty data field. The arrival of this type of packet gives the station a chance to transmit data if the station has any.

The entire ring module as a whole also has other inputs, namely,

- ***chip_enable*** - the signal to start the entire system.
- ***bus_clock*** - provides the clock signal for the station latches.
- ***system_clock*** - provides clock signal for transmitting module of each station.
- ***system_receive_clock*** - provides clock signal for receiving module of each station.
- ***preclear_generation_clock*** - provides clock signal for preclear signal generation components. The preclear signal is used in various modules in a station, e.g. counter.

- ***address_convert_clock*** - provides clock signal for the counter in the address conversion module of a station.
- ***Ethernet_address_input*** - a parallel input port of 48 pins which provides the Ethernet address of a station. These inputs are used at system start up to load the internal RAM with Ethernet address of each station on the ring. The Ethernet address information is used later on when converting Ethernet address to station address.
- ***station_address_input*** - a three pins input port that provides the station address information to the internal RAM at the system start up.
- ***ram_oe*** - provides the RAM output enable signal. This is a RAM I/O mode control input. A logical low on the input, together with a logic high on *ram_we*, enables reading of the RAM contents.
- ***ram_we*** - provides the RAM input enable signal. This is a RAM I/O mode control input. A logic low on this input enables writing into the RAM. A logic high together with a logic low on *ram_oe* enables the reading of the RAM contents.

3.2 Description of Transceiving Station

The station is the basic building block of the LAN. It provides the access to the data bus for the external data terminals. A station also controls the data flow on the data bus and inside the station through various control modules in the station. A block diagram in Figure 3-3 shows the overall structure of a station.

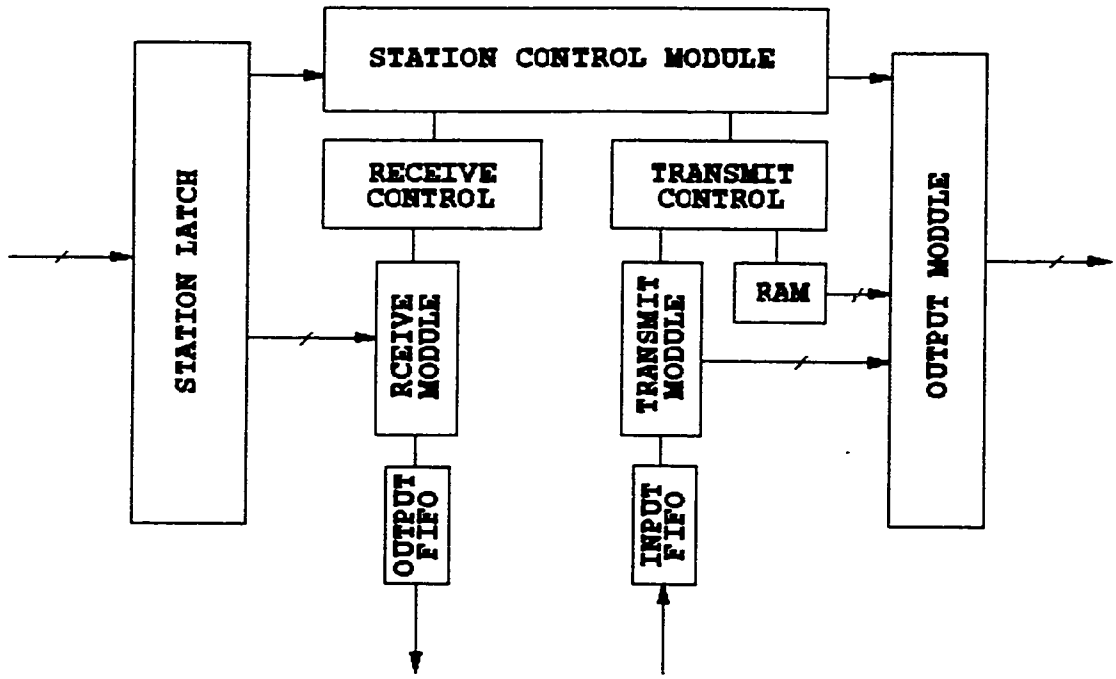


Figure 3-3 : structure of a transceiving station

The station latch stores the 33 bits of data, coming from the preceding station, on the parallel data bus at the rising edge of the bus clock. Data is stored at the latch for processing for the duration of a bus clock cycle.

The station control module decides, upon the arrival of a new data packet, if the packet is meant to be received by this station, or if the station can now insert a packet to be transmitted onto the data bus, or if the station should pass this incoming packet to the next station. The station control module controls the receive control module to start downloading data from the data bus, and controls the transmit control module to start uploading data. It also controls the output module to prepare the data to be clocked out on the next rising edge of the bus clock.

The receive module downloads the data from the data bus in parallel form and outputs the data to a FIFO (First In First Out) buffer in serial form. It is basically a parallel to serial converter.

The receive control model, consisting of counter and other control circuitry, controls the downloading and parallel to serial conversion process.

The transmit module takes the data from the input FIFO buffer, segments the data stream into segment of 24 bits. It also presents the data stream from the FIFO to the transmit control module to extract the destination station address information.

The transmit control module controls the segmentation process taking place in the transmit module. It also performs the address mapping function.

A RAM module is used to provide the necessary mapping between the Ethernet address of a DTE and the station address of the station the DTE being attached to. The contents of the memory is loaded at the system start up. When a new Ethernet packet comes in, the transmit control module compares the Ethernet address in the incoming Ethernet packet with the Ethernet address stored in the RAM and determines which station this Ethernet packet should be sent to.

The output module provides the final output of a station. This output is clocked out to the station latch of the next station at the rising edge of the next bus clock. Depending on the decision made by the station control module, the final output can be a real data packet, an Ack (acknowledgment) packet, a Nack (negative acknowledgment) packet or an empty packet.

3.3 The Operation of the LAN

In this section, the detailed operation of the proposed architecture is discussed. The proposed architecture is modular and scalable in the sense that each station is a self-contained entity. Hence, it is sufficent to focus our attention on the design of one station.

3.3.1 Station Control Module's Operation

A data packet is clocked into the station latch at the rising edge of the bus clock. After it has been processed by a station, the packet is sent to the output module and clocked out, to the station latch of the next station, at the next rising edge of the clock.

When a data packet is clocked in, the station control module examines the destination address field and the packet type indicator field. If the destination address of the incoming packet matches the station address of the station, the packet is subject to further processing. If the destination address of the incoming packet does not match the station address of the station, the station control module passes the packet from the station latch to the output module, and to the next station.

When the destination address of incoming packet matches the address of the station, the station control module examines the two bits packet type indicator field and determines the required action to be performed. If the incoming packet is a real data packet, the station control module will inform the receive control module, by sending a *receive* signal, to start the receiving process, provided the output FIFO is not full. Upon the successful reception of data by the output FIFO, the station control module tells the output module to send an Ack packet to the source station.

If the output FIFO is full before the receiving process starts, the station control module will tell the output module to sent out a Nack packet. If the output FIFO becomes full during the receiving process, the station control module will also tell the output module to

put a Nack packet on the data bus. In both cases, the Nack packet is sent to the source station. The source station will then resend the packet to this destination station.

If the incoming packet is an Ack packet, this means that a real data packet sent by this station has been successfully received by the destination. Therefore, the station control module will send a *send_empty* signal to the output module. This tells the output module to put an empty packet on the data. This empty packet can then be used by any station, on the data bus, who wants to transmit data. In the meanwhile, the station control module also tells the transmit control module to start loading next 24 bits of data into transmit module so that it will be ready for transmission when an empty packet arrives at the station.

If the incoming packet is a Nack packet, this means that a real data packet sent by this station has not been successfully received by the destination and the destination station requests for a resend. Therefore, the station control module requests the output module, by sending a *resend* signal, to take the existing output of the transmit module and put it on the data bus and have it ready for the next bus clock. One important thing to notice is that the transmit module, under the control of the station control module, will not load new data in, after the transmission of a real data packet, until an Ack packet is received by the station. This ensures that previously sent data is always ready for resend if required.

If the incoming packet is an empty packet, the station control module sends a *send* signal, to the output module, which tells the output module to take the real data packet that is waiting at the transmit module and put it on the data bus.

3.3.2 Receiving Data

When the receive control module receives the *receive* signal from the station control module, it clears the counter inside the receive control module. The counter starts counting and controls the receive module so that the data is sent to the FIFO serially.

The station control module monitors the output of the counter. If the FIFO becomes full before the counter finishes counting, this means that some data is not clocked into the FIFO properly, therefore part of the real data packet was lost. Then, the station control module requests a Nack packet to be sent to the transmitting station. On the other hand, if the FIFO is always ready to take more data during the entire counting process, the station control module knows that all 24 data bits have been correctly received by the FIFO buffer. After all data bits are transferred, the station control module requests an Ack packet to be sent to the transmitting station.

3.3.3 Transmitting Data

The transmit control module performs the following functions:

- detects the appearance of data in the input FIFO.
- detects the beginning of an Ethernet packet.
- controls the loading of the data from the input FIFO.
- determines the destination station address.
- segments the Ethernet data packet into 24 bits segment.
- assembles the real data packet that will be transferred on the data bus.

The transmit module is a long string of shift registers. At the system start up, all the shift registers are preset to ※0§ . With the control from the counter inside the transmit control module, the transmit module shifts in the data from the input FIFO buffer. The entire string of shift registers is divided into two sections, A and B, as shown in Figure 3-4. The first section, A, at the head of the string has 24 shift registers. They are directly involved in the assembly of real data packets transferred between stations. Section B facilitates some other functions besides shifting data, as will be explained in the following.

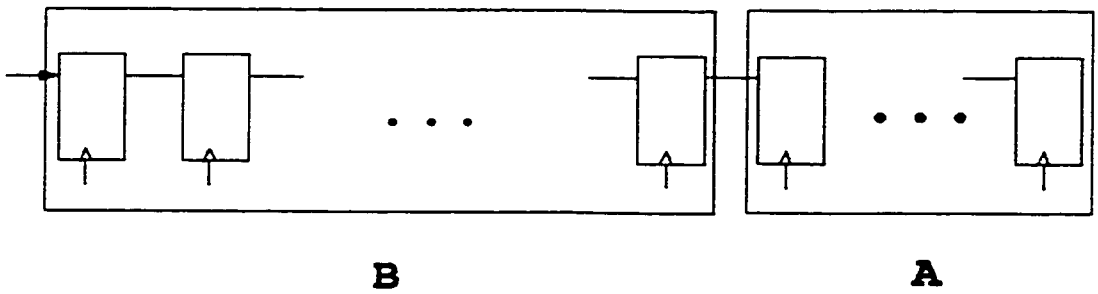


Figure 3-4 : shift register string

The format of Ethernet packet is shown in Figure 3-5.

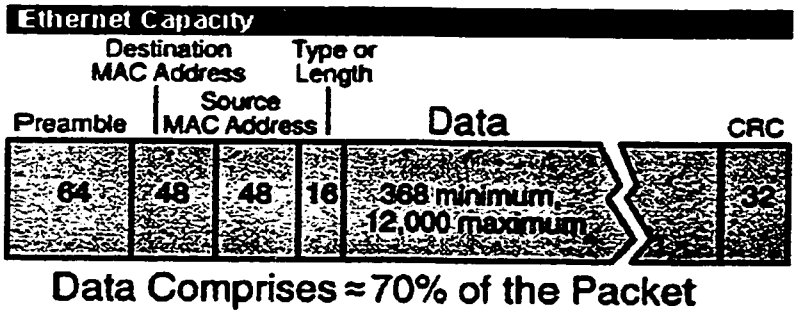


Figure 3-5 : Ethernet packet format

When data appears in the input FIFO buffer, the transmit control module starts shifting data into section B. The transmit control module looks for the preamble on the section B of shift register string. Once found, the transmit control module holds the new Ethernet packet and prevent it from being shifted into section A till the last several bits from the Ethernet packet precedes it has been sent off.

When the station receives an Ack packet, the transmit control module clears a counter. The counter starts counting and controls the shift registers to shift data from section B into section A. If this is the start of a new Ethernet packet, the transmit control module latches the 48 data bits, which is the destination DTE's Ethernet address, immediately

follows the preamble. Then, while keeps on shift data bits in, the transmit control module starts the address conversion process. It compares the Ethernet destination address with the memory contents in the RAM. Once a match is found, the corresponding station address is stored in registers and will be used to assemble the real data packet. By the time the data shifting action is completed, the address conversion has also finished. The 24 bits of data and the 3 bits of destination station address are ready for the assembly of the real data packet. These information is passed to the output module when the transmit control module receives the *send* signal from the station control module, i.e., an empty packet has been received. Over there, under the control of the station control module, other control bits are generated and the new real data packet is assembled and put on the data bus.

If the data bits from the section B is in the middle of a Ethernet packet. The data is simply shifted into section A. In this case, no address conversion is needed. The station address is still available from the registers storing the station address. The address conversion process only take place when a new Ethernet packet is shifted into to section A.

In both cases, the shift registers will hold the data that is sent out in case the data needs to be resend. As soon as the Ack packet comes back, the shift registers start shifting in another 24 bits of data in to section A and prepares for the next transmission.

Chapter 4

FPGA Implementation

The proposed LAN on a chip architecture is designed in VHDL (Very high speed integrated circuit Hardware Description Language) and is implemented on FPGA device. The target device is an Altera Flex10k FPGA. The overall architecture is shown in Figure 4-1.

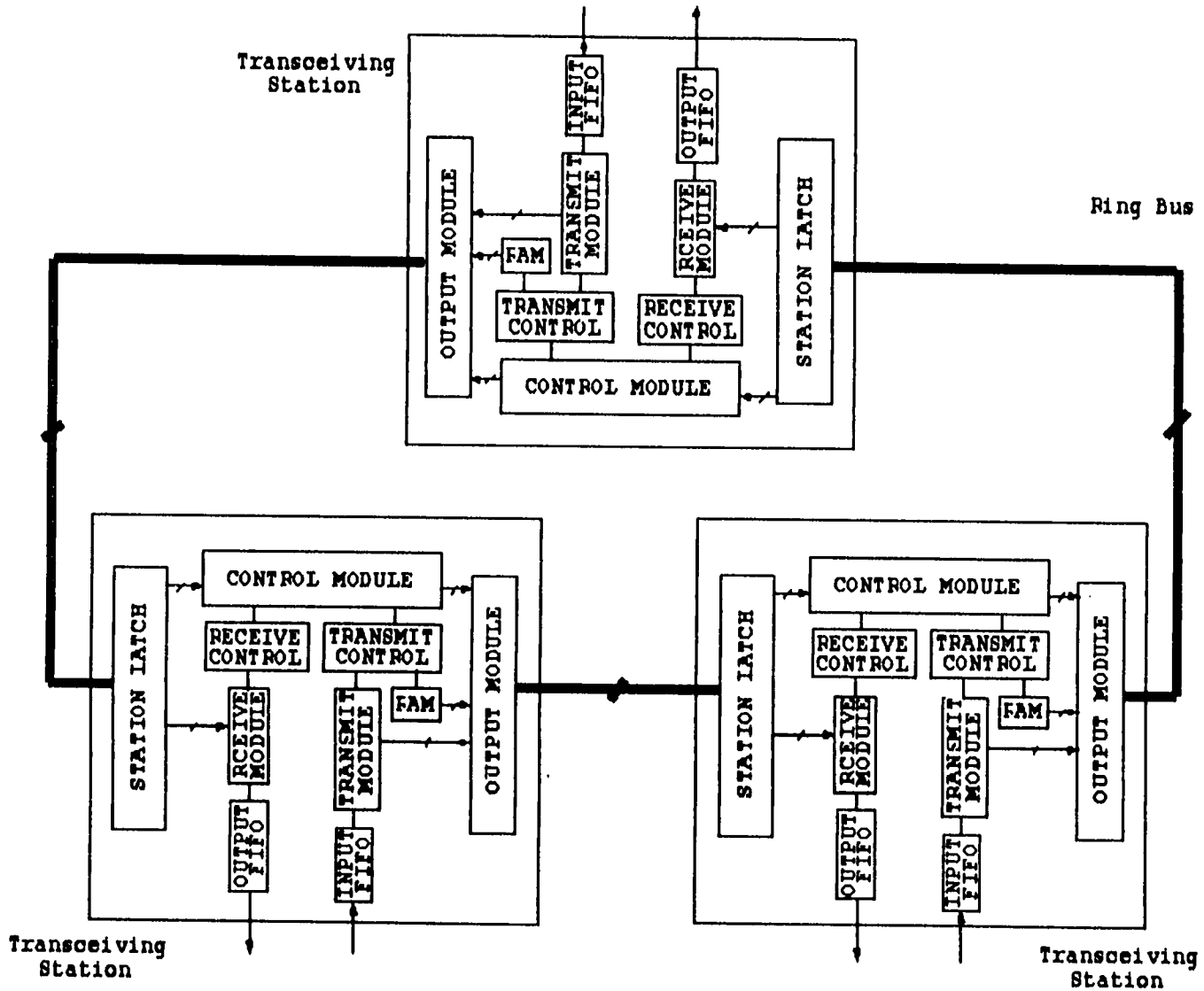

Transceiving Station
INPUT FIFO
OUTPUT FIFO
OUTPUT MODULE
TRANSMIT MODULE
RCEIVE MODULE
STATION LATCH
FAM
TRANSMIT CONTROL
RECEIVE CONTROL
CONTROL MODULE
Ring Bus
Transceiving Station
Transceiving Station

Figure 4-1

4.1 Station Latch

The diagram for station latch is shown in Figure 4-2. Station latch has one input bus, one output bus, a clock input and a power up reset input. The input and output bus are 33 bits wide which can accommodate the standard packet transferred on the data bus. The power up reset pin resets the station latch to all " 0" at system power up. The clock signal latches the data on the input bus at the rising edge of the clock. In the design, the input bus is connected to the input of the station while the output bus is connected to the inputs of the station control module and the output module. The power up reset pin takes in the reset signal at system start up and the clock pin is connected to the bus clock generation component inside the station control module.

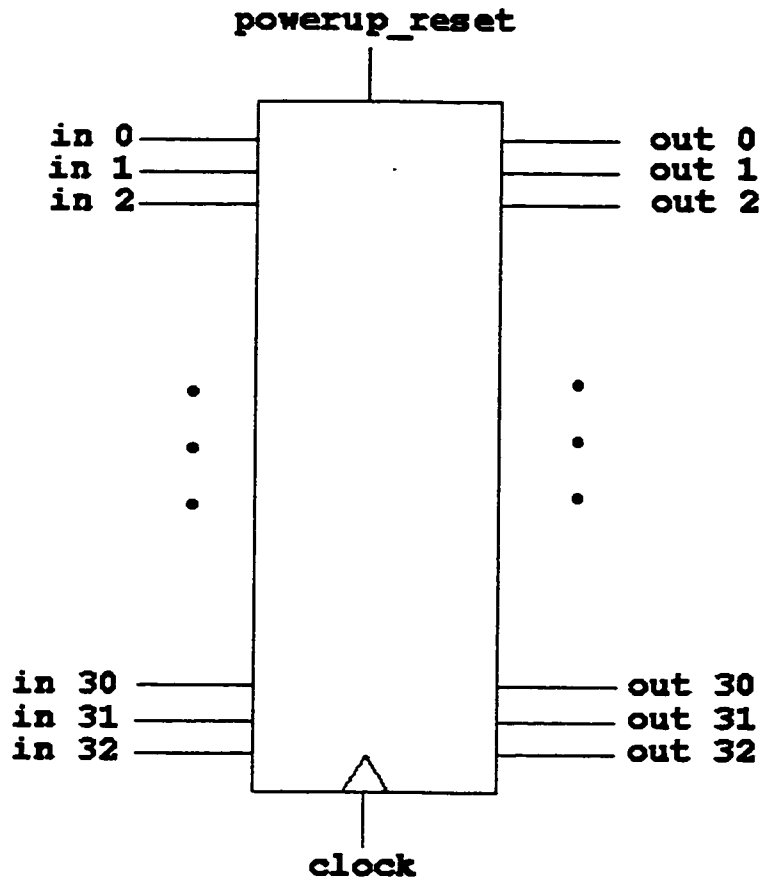


Figure 4-2 : station latch

4.2 Station Control Module

The station control module is composed of several components, including a bus clock generation component, a communications control component, a send signal control component, a receive signal control component and an Ack_Nack signal generation component.

Before we go into details of each of these component, we need to take a look at a generic component called preclear signal generation component that is used in various places in the design. This name was given since this component is designed and primarily used for this purpose. However, this generic design is also deployed for other purpose.

4.2.1 Preclear Signal Generation Component

The diagram of this preclear signal generation component is shown in Figure 4-3. The component has one input pin, I, one input enable pin, I_EN, one clock input and two outputs, Q and Q_N. The outputs are compliment of each other. While the input enable permits, this component monitors the input signal. Whenever there is a event happens on the input pin, the output Q gives a logic high ("1") for one clock cycle. An event is defined, in this implementation, as a logic high. In other words, the output Q gives a "1" for one clock cycle whenever the input I goes from "0" to "1".

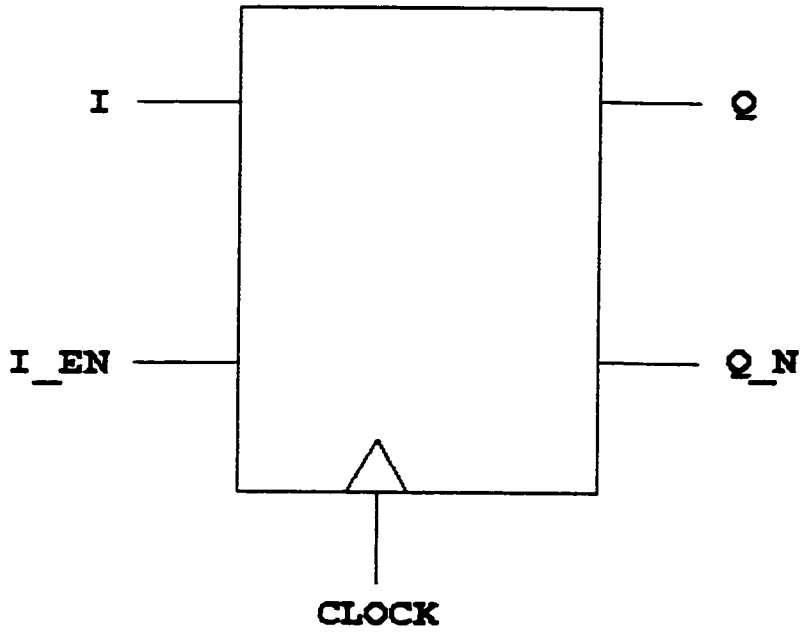


Figure 4-3 : preclear signal generation component

This component is designed as a finite state machine with two D flip-flops and some combinatorial logic circuit. The finite state machine has a state diagram shown in Figure 4-4. When the input I is " 0" , the output is " 0" and the finite state machine stays in state **b**. When the input changes from " 0" to " 1" , the output changes to " 1" at the first rising edge of the clock. The finite state machine goes to state **a**. If the input continues to be " 1" , the finite state machine stays in state **a** and produces an output of " 0" . When the input becomes " 0" , the finite state machine goes back to state **b** and gives an output of " 0" .

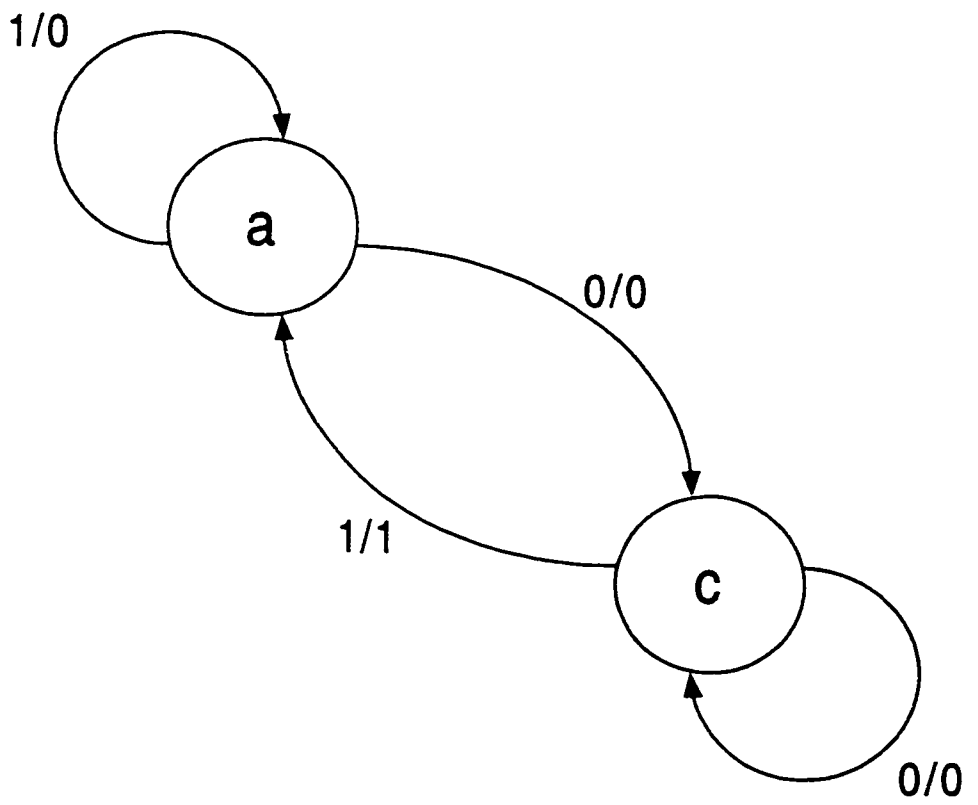


Figure 4-4 : state diagram for the preclear signal generation component

As mentioned above, this preclear signal generation component is mainly designed for generating preclear signal to clear the counter. In those cases, the *send* or *receive* signal are used as input to the preclear signal generation component, and the output clear the contents of the counter involved to zero and starts the counting process. However, the use of the preclear signal generation component is not limited to the above purpose only. In fact, it is very useful on many other places, which we will discuss later, as well.

4.2.2 Bus Clock Generation Component

The bus clock generation component is used to reproduce the bus clock signal. It is basically the preclear signal generation component. The input of the bus clock generation component is the external bus clock signal. The output Q feeds the clock signal to the station latch. The CLK input is connected to the preclear_generation_clock. The Q_N

output is an important enable signal used in other part of the station control module. There is no special features on this reproduced bus clock signal. It is a clock signal of the same frequency as the external bus clock signal, except the cycle duty has been change. The duration for clock high has been dramatically reduced. But this change does not affect the proper functioning of the station latch. However, one advantage of this bus signal reproduction is that the reproduced bus clock signal (Q) and the enable signal (Q_N) are always synchronized. This makes the enable signal a very useful control signal as we will see later.

4.2.3 Communications Control Component

The diagram of the communications control component is shown in Figure 4-5. This component is responsible for determining what type packet the incoming packet is and what action the station should take.

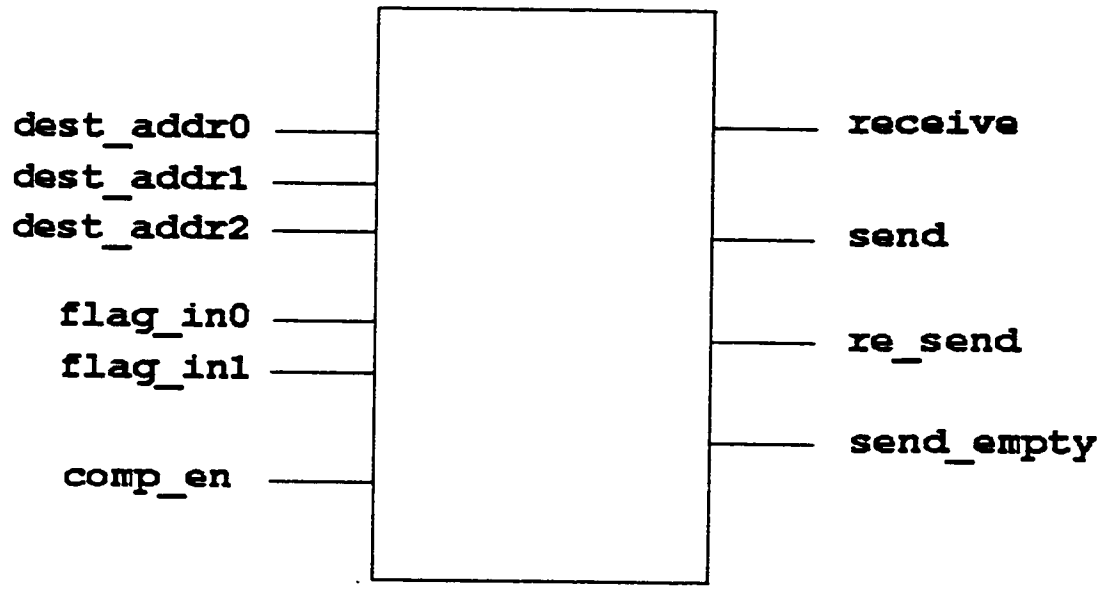


Figure 4-5 : communication control component

The communications control component has three sets of inputs. The three bits dest_addr input and the two bits flag_in input are connected to the output of the station latch. They take the data in the destination address field and the packet type indicator field, respectively. The comp_en input passes the enable signal produced by the bus clock generation component described above to the component. The receive output is connected to the receive control module and passes the receive signal to initiate receiving process. The send outputs is connected to the transmit control module and the output module to issue the send signal that initiates the transmitting process and guides the assembly process. The re_send output sends re_send signal to the output module to assemble the real data packet once again for retransmission. The send_empty output is connected to output module and passes the send_empty signal which ask the output module to put an empty packet on the data bus.

When the bus clock, generated by the bus clock generation component, clocks in the data, it takes a very short transient period for the data signals to settle down on the outputs of station latch. It is not desirable to reflect this transient period on the output of the communications control component. Since the *comp_en* signal is slightly delayed with respect to the bus clock signal, but enough to mask out the transient period, the *comp_en* signal ensures that stable signals are presented to the communications control component. When the *comp_en* signal is a logic low, all the outputs are " 0" . As soon as the *comp_en* signal goes to a logic high, the communications control component takes the inputs and does the calculation and presents the result at the outputs by setting one of the outputs to a logic high.

When the incoming packet' s destination address matches the station address of the current, the communications control component examines the packet type field. If a " 00" is found, this means the incoming packet is an empty packet, the station can transmit data if there is any. Therefore, the communications control component puts send output to logic high. If it is a " 11" , this indicates that the incoming packet is a real data packet for this station. Therefore, the communications control component then puts the

receive output to logic high. If the packet type field has the value " 10" , it means that the incoming packet is an Ack packet, the send_empty output goes to logic high. If the packet type field has the value " 01" , this means that the incoming packet is a Nack packet, then the resend output goes to logic high.

Actions are then taken by other components to control the activities in the station. The *receive* and *send* signals are passed to send signal control component and receive signal control component, respectively, where other conditions are also examined before initiating the receiving and transmitting process. The *send_empty* and *resend* signals are passed to output module directly to direct the output module to put the appropriate data on the data bus.

4.2.4 Send Signal Control Component

The diagram for the send signal control component is shown in Figure 4-6. The send signal control component makes the final decision on if a send signal should be generated and passed to the transmit control module, which directly control the transmitting process.

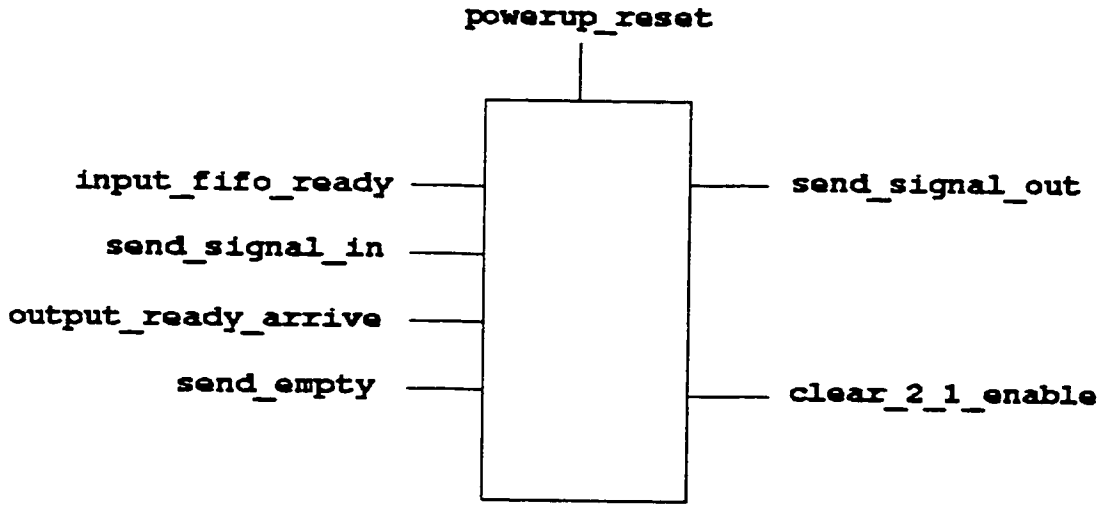


Figure 4-6 : send signal control component

The send signal control component have five inputs and two outputs. The powerup_reset input signal resets the component at system power up. The input_fifo_ready input signal indicates whether there are data to transmit. The send_signal_in input takes the send output from the communications control component. The output_ready_arrive input signal indicates whether new data has arrived since the last time the input FIFO has been emptied. The send_empty input is connected to the send_empty output from the communications control component. The send_signal_out output gives the send signal to the output control module to put the data on the bus. The clear_2_1_enable output signal is employed in the transmit control module to control the counter. How the clear_2_1_enable signal is used will be discussed later.

A transmitting process can be triggered by either the *send_empty* signal or the *output_ready_arrive* signal. The triggering happens under two circumstances. The first circumstance happens when the data bits are continuously available for transmission, i.e. there are still data held by the shift registers in the section B of the long register string inside the transmit module (see Figure 3-4). In this case, when the *send_empty* signal is presented to the send signal control component, it means an Ack packet has been received by the station. The last transmission was successful. Since there are still data to be sent, the station can start preparing for the next transmission by shifting 24 data bits form section B to A, knowing there is no need to keep the last 24 data bits for retransmission any more. When the next empty packet is received by the station, the communications control component sends a *send* signal to send signal control component. Upon receiving the *send* signal, the send signal control component sends out the *final send* signal. The condition for sending out the *final send* signal is a *send_empty* signal has to be received after the last transmission. This ensures that the data is available for retransmission if needed. If empty packets arrive before the Ack packet, the send signal control component just ignore them and no *final send* signal will be issued. In other words, the communications control component identifies the empty packets, but it is up to the send

signal control component to decide when to transmit, even though, in theory, all incoming empty packets make the bus clock cycle available for the station to transmit.

The second situation takes place when data becomes available again in the input FIFO after the input FIFO being emptied up due to previous transmissions, or after the system start up where the FIFO was originally empty. In these cases, the reception of Ack packet is no long a valid triggering event for the transmit module to start shifting in data bits from section B to section A. In the system start up case, the Ack packet won' t even appear. In these situations, the arrival of data bits in section B of the shift register string in transmit module is used to trigger the shifting data bits into section A and prepare for a new transmission. When the transmit module reads the data from the input FIFO, it shifts in both the data bits and the output ready signal at the same time. The output ready signal is shifted in using a separate shift register string which has the same length as section B. When the output ready signal arrives at the head of its string, an *output_ready_arrive* signal is generated and sent to the send signal control component. The *output_ready_arrive* signal is also used inside the transmit module to clear the counter and start shifting data bits from section B into section A. This *output_ready_arrive* signal has the same effect, to the send signal control component, as the *send_empty* signal in the first situation described above. When the next empty packet arrives at the station, the send signal control component knows this bus clock cycle can be used to transmit data, therefore a *final send* signal is sent to output module.

The triggering of shifting data into section A is done by clearing the counter to zero. The counter can be cleared by the preclear signal clear_2_1 or clear_2_2. The clear_2_1 signal is generated from the *send_emtpy* signal issued by the communications control component. The clear_2_2 signal is generated from the *output_ready_arrive* signal. Normally, the clear_2_1_enable is set to " 1" . In the second situation, the clear_2_1_enable output is set to " 0" when the output_ready_arrive signal is received. But the following falling edge of the send_signal_in will set the clear_2_1_enable back to

" 1" . This allows one of the two preclear signal generation components to be disabled and let the other one control the initiation of the transmitting process.

4.2.5 Receive Signal Control Component

The diagram of the receive signal control component is shown in Figure 4-7. As with the send signal control component, the receive signal control component makes the final decision on initiating the process of receiving.

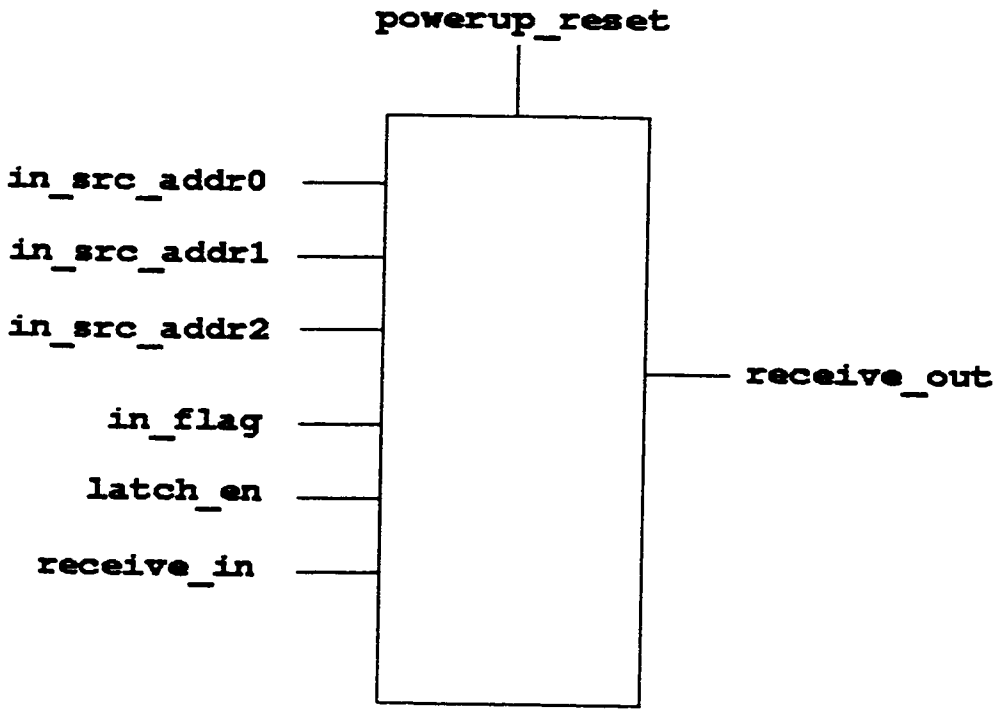


Figure 4-7 : receive signal control component

The receive signal control component has seven inputs and one output. A three bit in_src_addr input reads the source station address from the station latch' s source address field. The in_flag input is connected to the group indicator output pin on the station latch. the latch_en input takes in the enable signal from the bus clock generation

component. The powerup_reset input signal resets the component at system power up. The receive_in input takes in the receive signal from the communications control component. The receive_out output sends out the final receive signal to receive control module.

When a real data packet reaches the destination station, the communications control component identifies the packet and issues a receive signal. The receive signal control component takes this receive signal and decides whether the station really wants to receive this real data packet. The receive signal control component has internal memory for the source station address and the group indicator of the incoming data packet. At system start up, this internal memory is cleared. Every time a receive signal is received, the receive signal control component checks the group indicator in the incoming data packet against the group indicator information stored in the internal memory. If the stored group indicator has a value of " 0" , it tells the receive signal control component that there is no ongoing transmission of Ethernet data packet to this station. In other words, this station is not in the middle of receiving a complete Ethernet packet and the station is ready to receive a new Ethernet data packet. If the incoming data packet has a group indicator of value " 1" and the stored group indicator is of a value " 0" , the receive signal control component knows that the first 24 bits of a new Ethernet data packet arrives at this station. The receive signal control component stores the new source station address into the internal memory and change the stored group indicator to " 1" . It also send out the final receive signal to receive control module to start the receiving process.

All real data packets come with a group indicator of value " 1" , except the last packet of the group, which contains the last several bits of the entire Ethernet data packet. The last real data packet has a group indicator of value " 0" , indicating the end of an Ethernet data packet.

When the subsequent real data packet comes in, the receive signal control component compares the group indicator values from the incoming packet and the internal memory.

If both values are " 1" , the receive signal control component makes further examination by checking the source station address of the incoming data packet against the one stored in the internal memory. If the source station addresses are the same, the receive signal control component knows that the incoming data packet is part of the same Ethernet data packet sent by the same source station. Therefore, a final receive signal is send to the receive control module.

If the source station address of the incoming real data packet is not the same as the stored source station address, this tells the receive signal control component that another station on the data bus is trying to send data to this station. However, since this station has not finished the reception of the current Ethernet data packet, it can not start receiving the next one. Therefore, the receive signal control component ignores the receive signal sent out by the communications control component and does not send the final receive signal to the receive control module. In this case, the communications control component sends out a send_nack signal to output module.

When the incoming real data packet has a group indicator of value " 0" , the receive signal control component knows the end of the Ethernet data packet the station has been receiving has been reached. Hence, the receive signal control component changes the stored group indicator value from " 1" to " 0" . This allows the receive signal control component to get ready for the reception of the next Ethernet data packet. When the next real data packet, which can only be the first 24 bits of an Ethernet data packet, comes in, the receive signal control stores the new source station address in the internal memory and repeat the procedure described above.

4.2.6 Ack_Nack Signal Generation Component

Ack_Nack signal generation component is used to generate the send_ack signal and send_nack signal. These signals control the output module to put Ack packet or Nack packet onto the data bus.

Figure 4-8 shows the ports on the Ack_Nack signal generation component. The component has five inputs and two outputs. The bus_clk input takes the bus clock signal from the bus clock generation component. The rec_en0 input is connected to the receive output of the communications control component. The receive_enable input takes in the final receive signal from the receive signal control component. The fifo_inready input is connected to the IR (input ready) output of the output FIFO buffer. The fifo_inready signal indicates whether the output FIFO is ready to take in more data. The counter_stop input is connected to the counter inside the receive control module. The counter_stop signal tells the Ack_Nack signal generation component if the counting process has completed. The send_ack output and send_nack output are connected to output module and controls the output module to send out Ack packet or Nack packet.

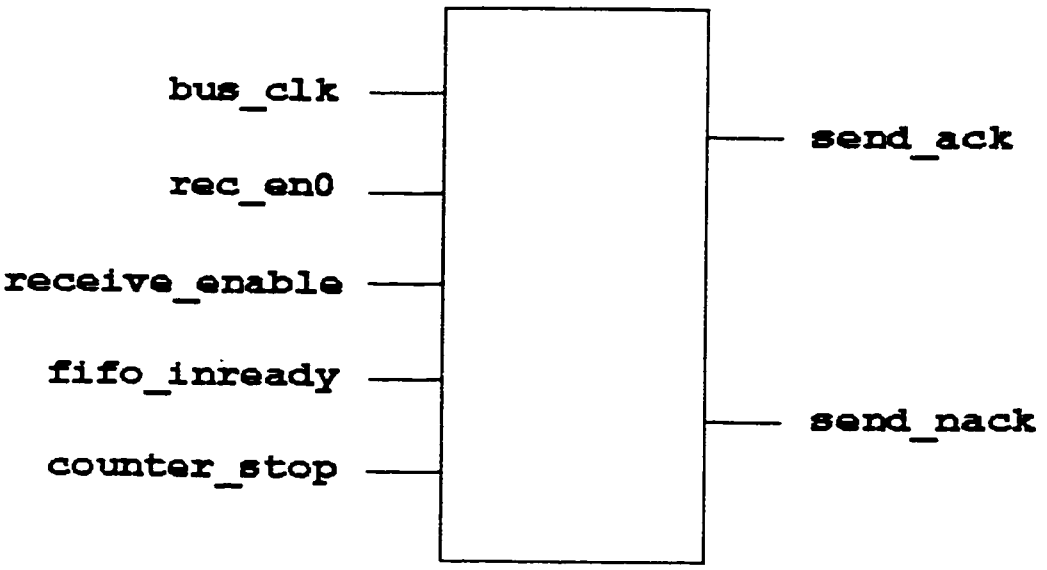


Figure 4-8 : Ack_Nack signal generation component

When a real data packet arrives at the destination station, the receiving process begins. Depending on whether the reception is successful or not, either an Ack packet or a Nack

packet has to be sent out. A successful reception is always assumed by the Ack_Nack signal generation component. Therefore, a send_ack signal is always generated, by setting the send_ack output to " 1" , as the receiving process starts. In the meantime, the send_nack output is set to " 0" . During the bus clock cycle where a receiving process takes place, either the send_ack output is set to " 1" and the send_nack output is set to " 0" or the other way around. When there is no receiving activities, both send_ack and send_nack outputs are set to " 0" .

The Ack_Nack signal generation component monitors the input ready output of the output FIFO as soon as the receiving process begins. If at any time the output FIFO indicates that it can not take in any more data, the Ack_Nack signal generation component sets send_nack output to " 1" and the send_ack output to " 0" . The send_nack signal stays on for the rest of the receiving process. Therefore, when the next bus clock comes along, a Nack packet is sent out. If all 24 bits of data was able to be shifted into the output FIFO, the send_ack will not be changed. Therefore, an Ack packet is sent out at the next rising edge of bus clock.

4.3 Receive Module

The receive module is used to convert parallel data bits, in the station latch, and send the data bits serially to the output FIFO. The receive module accomplishes this action under the control of the receive control module. The receive module is consists of one parallel-to-series component.

4.3.1 Parallel-to-series Converter

The parallel-to-series converter is similar to a multiplexer. It has 24 data inputs, one data output, one output enable input and five select control inputs. The data input pins are connected to the data bits outputs of the station latch. The data output of the parallel-to-series converter is connected to the data input of the output FIFO. The output enable input is connected to the input ready output of the output FIFO. The select control inputs are

connected to the COUNTER_OUT outputs on the five-bit counter inside the receive control module. During the counting process, the COUNTER_OUT outputs provide the count, 1 to 24. This count information is used as sellect control to put appropriate input data bit on the output. A diagram of the parallel-to-series converter is shown in Figure 4-9.

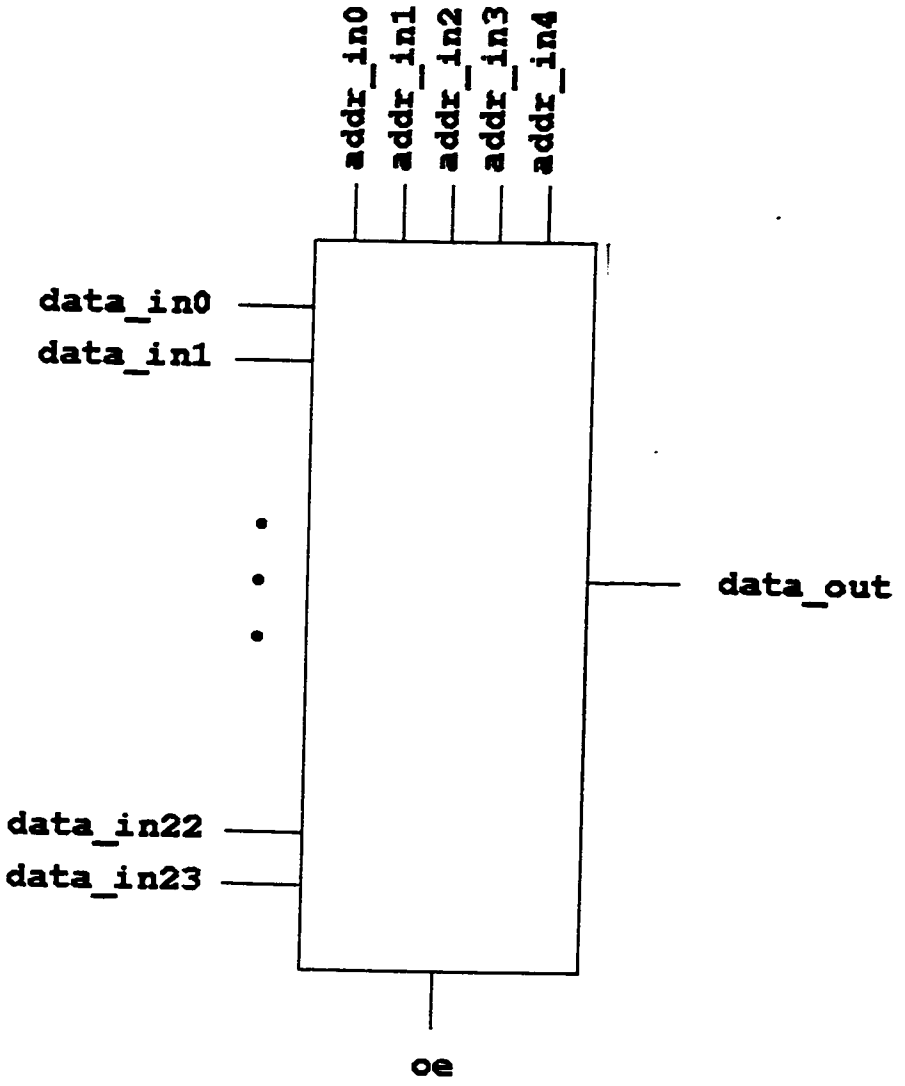


Figure 4-9 : parallel-to-series converter

This is a simple approach to accomplish the parallel-to-series conversion. However, due to the technology of the target device, the delay from applying a select control input to getting the corresponding output is about 17 ns, which is relatively big comparing to the 20 ns duty cycle for the system_receive_clock. This requires the synchronization between the data output of the parallel-to-series converter and the write clock of the output FIFO. This problem is discussed later in this chapter.

4.4 Receive Control Module

The receive control module is responsible for controlling the receiving process once the process is initiated by the station control module. The module consists of a preclear signal generation component and a five-bit counter.

4.4.1 Preclear Signal Generation Component

The preclear signal generation component used in the receive control module is the same component described in section 4.2.1. However, in this module, the preclear signal generation component is connected differently so that it causes the counter to start the receive clock when a *receive* signal is sent out by the station control module.

A diagram of the preclear signal generation component is shown in Figure 4-10. Here, the component' s *receive_in* input is connected to the receive_out output of the receive signal control component in the station control module. The *enable* input is connected to the Q_N output of the bus clock signal generation component and it acts as an enable pin for the preclear signal generation component. The clock input pin takes in the preclear_generation_clock signal. The preclear_generation_clock is faster than other clocks in this design. This ensures quick response from the counter when control signal (final receive) is issued. Only one output pin, Q, is used in this case. The output signal clear_1 is sent to counter from this pin.

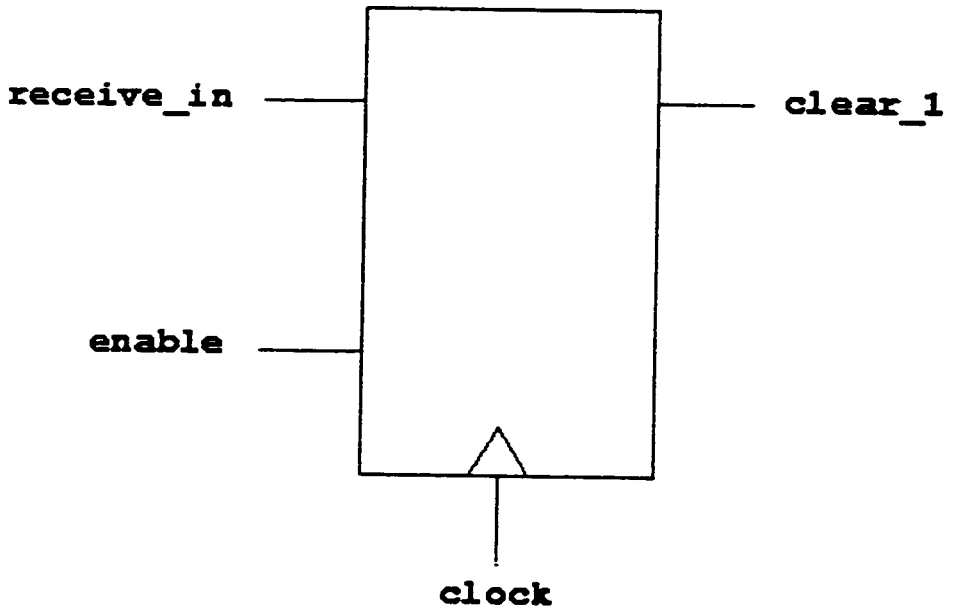


Figure 4-10 : preclear signal generation component in receive control module

When the rising edge of the bus clock loads a new packet into the station latch, the station control model starts examining the newly loaded packet and making decision. It takes some time for the outputs of the station control module to settle down. Hense, transient responses may show up at the station control module outputs. The enable signal provides a way to shield the preclear signal generation component from these possible transient responses and ensures that the component only responds to stable control signal input.

The preclear signal generation component monitors the receive signal output of the station control module. When the final receive signal is received, a clear signal is sent by setting the clear_1 to " 1" for one clear generation clock cycle. The clear signal is used by the five-bit counter to clear the counter to zero and starts the counting process.

4.4.2 Five-bit Counter

The five-bit counter is a key control component in both the receive control module and the transmit control module. It controls the clock used in the receiving and transmitting processes.

Figure 4-11 shows the diagram of the five-bit counter. The five-bit counter has three inputs and three outputs. The CLOCK input provides the counter with the system_receive_clock as its clock input. The PC input is connected to the clear_1 output of the preclear signal generation component. The POWERUP_RESET input receives the reset signal at system power up. The CLOCK_OUT output sends out the output clock signal to the receive module. The COUNTER_STOP output is connected to the Ack_Nack signal generation component and sends out a signal indicating the end of the counting process. This signal is used by the station control module for control purpose. The five-bit COUNTER_OUT bus outputs the counts of the counter. The counts are used as the address control for the parallel to series converter in the receive module.

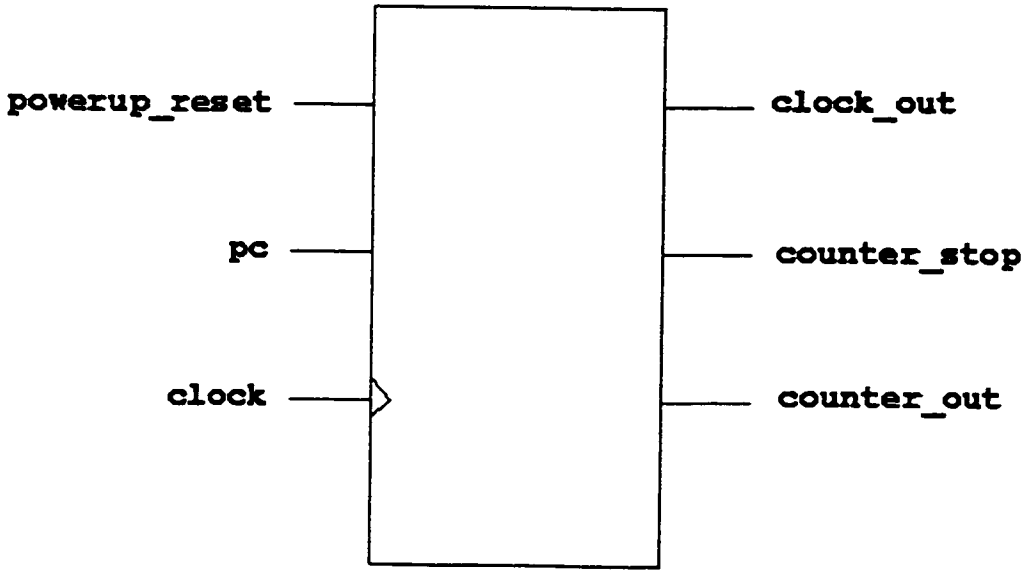


Figure 4-11 : five-bit counter

The five-bit binary counter was designed as a finite state machine. There are 28 states, state 0 through state 27, in the design. The CLOCK_OUT is enabled only from state 3 to state 26. These 24 clock cycles shift the 24 data bits in the receive module to output FIFO. The extra states are used for control purpose. Each state has a corresponding count output as listed in Table 4-1.

STATE #	COUNT OUTPUT
State 0	00000
State 1	00001
State 2	00011
State 3	00010
State 4	00110
State 5	00111
State 6	00101
State 7	00100
State 8	01100
State 9	01101
State 10	01111
State 11	01110
State 12	01010
State 13	01011
State 14	01001
State 15	01000
State 16	11000
State 17	11001
State 18	11011
State 19	11010
State 20	11110
State 21	11111
State 22	11101
State 23	11100
State 24	10100
State 25	10101
State 26	10001
State 27	10000

Table 4-1 : state assignment for five-bit counter

The assignment of binary count output value to each states does not follow that used for conventional binary counter. In a conventional binary counter, situation may exist where more than one bit needs to change simultaneously between neighboring states. For example, the output of count 3 and 4 would be assigned a binary value of " 00011" and " 00100" . In this case, three bits are required to change at the same time. Due to different delays experienced by different gates, racing condition may occur where some bits may change faster than others. Hence, some count output for other state may appear between state 3 and state 4. This is highly undesirable in this design since the count output of the five-bit counter component is used as address control for the receive module and the appearance of count output of other state, between two consecutive states, causes the data bit out of sequence when clocked into the output FIFO.

In this design, instead of a straight forward binary value assignment for count output of each states, the count output is assigned to each state in such way that adjacent states have count output that varies by only one bit. This scheme avoided the racing condition in the finite state machine, therefore, eliminated the unstable transient response and allows the five-bit counter always presents a series of stable count outputs at the COUNTER_OUT output port and also avoided providing the counter_stop signal at the wrong time.

Before the clear signal comes in, the five-bit counter is in state 27 and has a count output of " 10000" . The COUNTER_STOP, in the meantime, is set to " 1" , indicating that the counter has been stopped. When the clear signal is received, the counter goes to state 0 and clears the count output to " 00000" . This action changes the COUNTER_STOP to " 0" and enables the input system_receive_clock. The input clock signal is the event that the five-bit counter counts. Whenever the rising edge of the system_receive_clock appears, the counter counts 1 and changes from one state to the next one and the count output changes accordingly. The count output is presented to the receive module which puts the 24 data bits, from the data bus, at its output one at a time in the correct sequence. When the 27^{th} state is reached, the input clock is disabled and the COUNTER_STOP is set, again, to " 1" .

During the counting process, the output clock is enabled from state 3 to state 26, while the COUNTER_OUT is enabled from state 2 to state 25. The output clock follows the input clock and it is used as the write clock for the output FIFO buffer. The reason that the output clock is enabled one state, which is one system_receive_clock cycle, after the COUNTER_OUT is enabled is that the receive module has some delay for parallel-to-series conversion. As mentioned the section 4.3.1, this delay is about one system_receive_clock cycles. If the COUNTER_OUT and the clock output were issued at the same time, the data signal would not be ready at the data output of the receive module when the first system_receive_clock cycle arrives. By starting sending out clock output signal one clock cycle later, the data output and the write clock are synchronized, therefore proper write operation is maintained, i.e. exactly 24 data bits are written into output FIFO buffer in sequence.

4.5 Transmit Module

The transmit module is used to send data bits in the input FIFO buffer onto the ring type data bus. It provides the series-to-parallel conversion function while facilitating other features used for other control and processing purposes. The transmit module is composed of three components, namely, the series-to-parallel converter, the data shift component and the output_ready signal shift component.

4.5.1 Series-to-parallel converter

The block diagram for the series-to-parallel converter is shown in Figure 4-12. The series-to-parallel converter is, basically, a 24 bit long shift register. It is the section A of the shift register string described in section 3.3.3. It has one data input and one clock signal input. The data input takes in the data from the data shift component. It has 24 data outputs, which are connected, through the output module, to the ring type data bus. The clock signal is the system_clock under the control of the transmit control module.

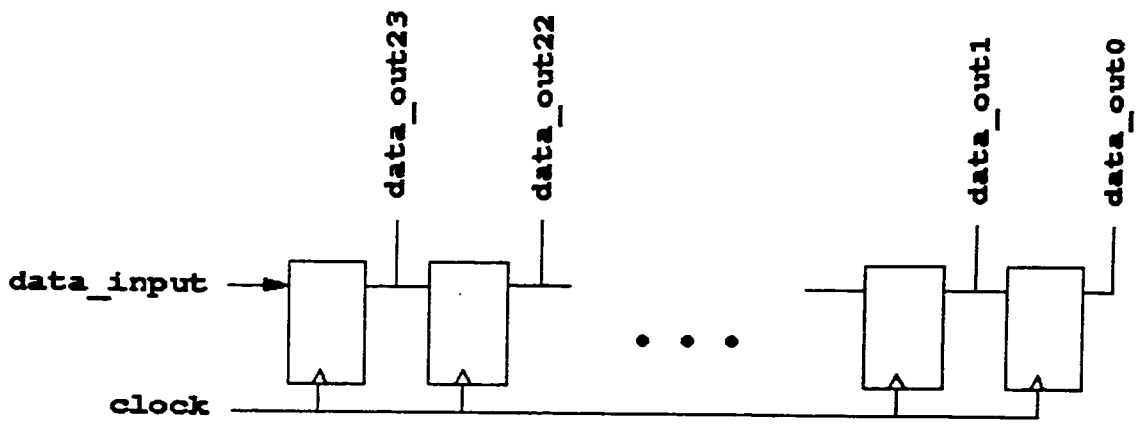


Figure 4-12 : series-to-parallel converter .

4.5.2 Data Shift Component

The data shift component is used to shift data bits in the input FIFO buffer into the station and make the data bits available for the transmission process. A diagram of the data shift component is shown in Figure 4-13. The data shift register consists of 116 D flip-flops. It is basically the section B of the shift register string described in section 3.3.3. The data shift component has one data input and one clock input. The data input is connected to the data output of the input FIFO buffer through the transmit control module. The clock signal is also being controlled by the transmit control module. Each D flip-flop provides an output, which gives a total of 116 outputs. These outputs are used for detecting the preamble in the Ethernet data packet and latching the MAC destination address in the Ethernet data packet header.

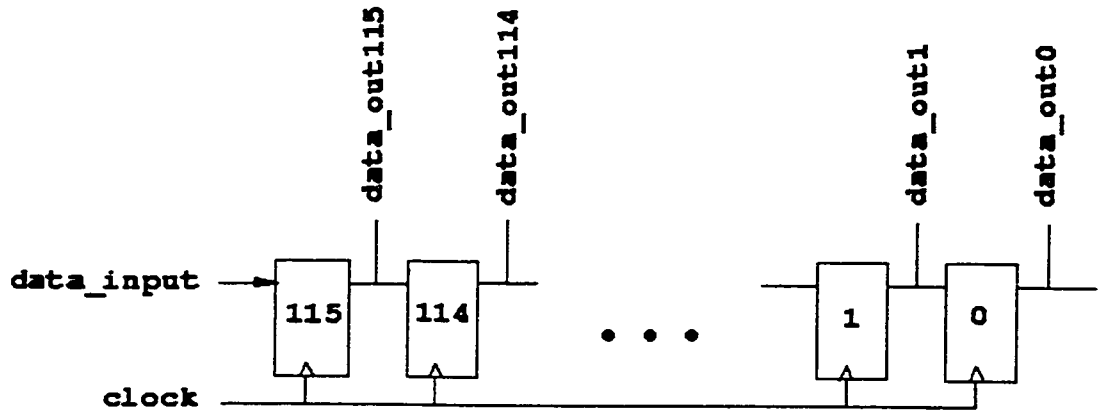


Figure 4-13 : data shift component

The output 0 to output 63 are connected to the preamble identifier component inside the transmit control module. When the Ethernet packet preamble reaches this part of the data shift component, the preamble identifier recognizes the preamble and triggers other control activities. The output 64 to output 111 is connected to the MAC address latch inside the MAC address converter module. The output of the latch is used for converting the MAC destination address to the station address. There are five extra shift registers also included in this shift register string. This gives some room for fine tuning for synchronization at system integration phase.

4.5.3 Output Ready Signal Shift Component

The output ready signal shift component is used to shift the output ready signal, from the input FIFO buffer, along with the data bits. The output ready signal is needed for controlling the transmit process. The diagram of the output ready signal shift component is shown in Figure 4-14. This component has one data input, one clock input and one data output. The data input is connected to the OR (Output Ready) port of the input FIFO buffer. The clock signal is controlled by the transmit control module. It is the same clock

signal as that in the data shift component. The output is connected to the shift control component inside the transmit control module.

Figure 4-14 : output ready signal shift component

The output ready signal shift component is made up of 116 D flip-flops, which is the same as with the data shift component. This ensures that the data bits and the output ready signal are in synchronization. The combination of the data shift component and the output ready signal shift component is equivalent to extend the input FIFO buffer into the station, however provides critical control information, e.g. the preamble and the MAC destination address, to the station.

4.6 Transmit Control Module

The transmit control module provides the key control function for the transmit process. It controls the shifting of data bits in various situations and it also controls the MAC address conversion. The transmit control module is composed of a five-bit counter, a shift control component, a preamble identifier component, an input control component, and two preclear signal generation components. The module takes inputs from the station control module and the transmit module and provides outputs to control the transmit module and the MAC address converter module.

4.6.1 Five-bit counter

The five-bit counter is the same design as the one used in the receive control module. Here, the CLOCK input is connected to the system_clock, while the CLOCK_OUT port sends the output clock signal to the shift control component. The PC port is connected to the send signal generation component in the station control module. When the final send signal is received, the counter is cleared to zero and the counting process starts. The POWERUP_RESET input takes in the powerup_reset signal which resets the counter at system power up. The COUNTER_STOP and COUNTER_OUT outputs are connected to the shift control component which controls the transmitting process.

The five-bit counter starts the counting process when a preclear signal is received and enables the output clock to follow the system clock for 24 clock cycles. The output clock signal is used to shift 24 data bits from the data shift component to the series-to-parallel converter.

4.6.2 Shift Control Component

The shift control component controls the shifting activity. The diagram of the shift control component is shown in Figure 4-15. The component has six inputs and two outputs. The fifo_or input takes the output of the output ready signal shift component in the transmit module. This signal tells the shift control component if there is data to shift into the series-to-parallel converter. The sys_clock input and the clock inputs take in the system clock and the counter output clock, respectively. It also inputs the count and the counter_stop signal from the five-bit counter in the transmit control module. The shift_stop input is connected to the preamble identifier component. This signal tells the shift control component if an Ethernet preamble has been found in the data stream. The clk_1 clock output is connected to the series-to-parallel converter and provides the clock signal to it. The clk_2 clock output is connected to the data shift component and provides the clock signal for the shift registers.

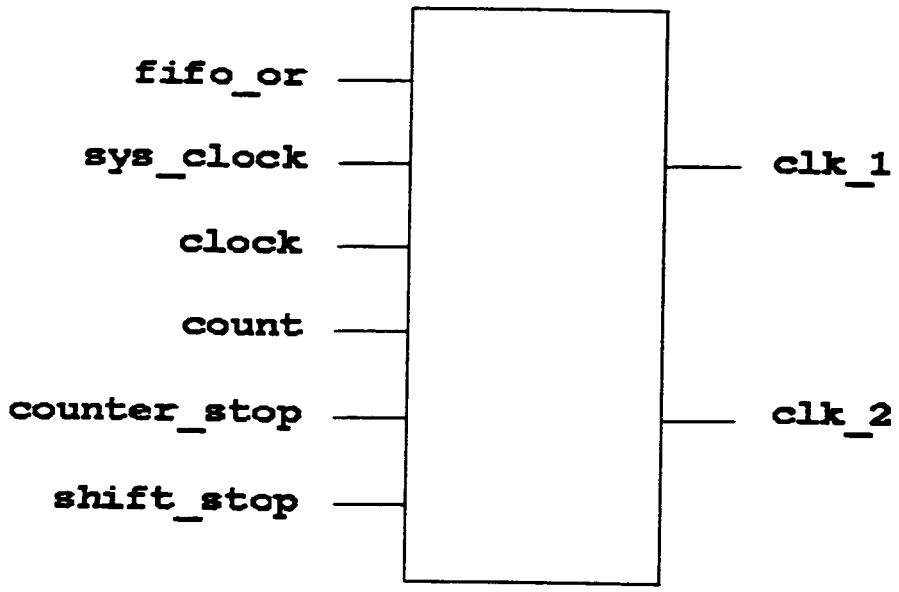


Figure 4-15 : shift control component

Generally speaking, there are three situations that needs to be considered in order to implement shift control functions properly. The first situation is when there is no output ready signal presented at the output of the output ready signal shift component, i.e. the shift control component receives a " 0" at the fifo_or input. This may happen when the station has not received its first outgoing Ethernet packet from the DTE, yet. Or, the succeeding Ethernet packet does not immediately follow the preceding Ethernet packet. In both cases, the entire data shift component or some part of it is not occupied by useful data. Normally, these space is set to " 0" . But, by looking at the output ready signal at the output of the output ready signal shift component, the shift control component knows the data output 0 of the data shift component does not carry a meaningful data bit. In this situation, the shift control component enable the clk_2 output and disable the clk_1 output. The clk_2 follows the system_clock and provides the clock signal to the data shift component and the input FIFO buffer. This causes the data shift component to keep on shifting in data until the output ready signal appears at the output of the output ready

signal shift component. In the meanwhile, the first bit of an Ethernet packet also appears at the output 0 of the data shift component. This design ensures that all data packets sent onto the data bus contains real data.

The second situation happens when the station is in the middle of sending an Ethernet packet. In this case, the output ready signal is " 1" , indicating there is real data in the data shift component waiting to be sent. When the final send signal is received, the counter sends out 24 clock signal. The data shift component enables both clk_1 and clk_2 outputs to pass this clock signal to the series-to-parallel converter component, the data shift component and the input FIFO buffer. Both series-to-parallel converter and data shift component shift in 24 bits of data synchronously. When the counting process stops, the series-to-parallel component is ready for passing the 24 bits of data to the output module for data packet assembly.

The third situation takes place when the tail of an Ethernet packet is reached. The length of an Ethernet packet is in number of bytes. That is to say the last real data packet may or may not be fully occupied by real data bits. It may contains 24 bits, 16 bits or 8 bits of real data. The unused data field needs to be padded with " 0" or " 1" . When the counter starts the counting process after the final send signal is received, as in the second situation, both clk_1 and clk_2 are enabled. If the Ethernet packet is not immediately followed by another Ethernet packet, " 0" s are shifted into the series-to-parallel converter component from the data shift component, since, in the data shift component, " 0" s are stuffed between two Ethernet packets. If the tail of the preceding is immediately followed by another Ethernet packet, the preamble will be detected by the preamble identifier component which informs the shift control component the arrival of the new Ethernet packet by setting shift_stop input to " 0" . At this moment, the data shift component checks the counter' s count output. If the counter still has not quite finished the entire counting process, i.e. less than 24 bits has been shifted into the series-to-parallel converter component, the data shift component disables the clk_2 and keeps the clk_1 enabled. This stops the shifting action in the data shift component and avoids

that the head of another Ethernet packet being sent with the old Ethernet packet. Since the two Ethernet packets may not go to the same station. In the meantime, the series-to-parallel converter component keeps on shifting in data. Since the data shift component has been stopped, the first bit of the Ethernet preamble, which is " 1" , is shift in to fill the unused data field following the last bit of the preceding Ethernet packet.

4.6.3 Preamble Identifier Component

Each Ethernet packet starts with the preamble which consists of two bits of " 1" and followed by 62 bits of alternating " 1" and " 0" . The preamble identifier component is used to monitor the data stream shifted in by the data shift component in order to find the Ethernet preamble, hence the beginning of a new Ethernet packet. A diagram of the preamble identifier component is shown in Figure 4-16. The component has 64 inputs which are connected to the output 0 through 63 of the data shift component. It has one output which gives a " 0" when an Ethernet preamble is found. Otherwise, the output stays at " 1" . The output is connected to the shift control component and the MAC address conversion module.

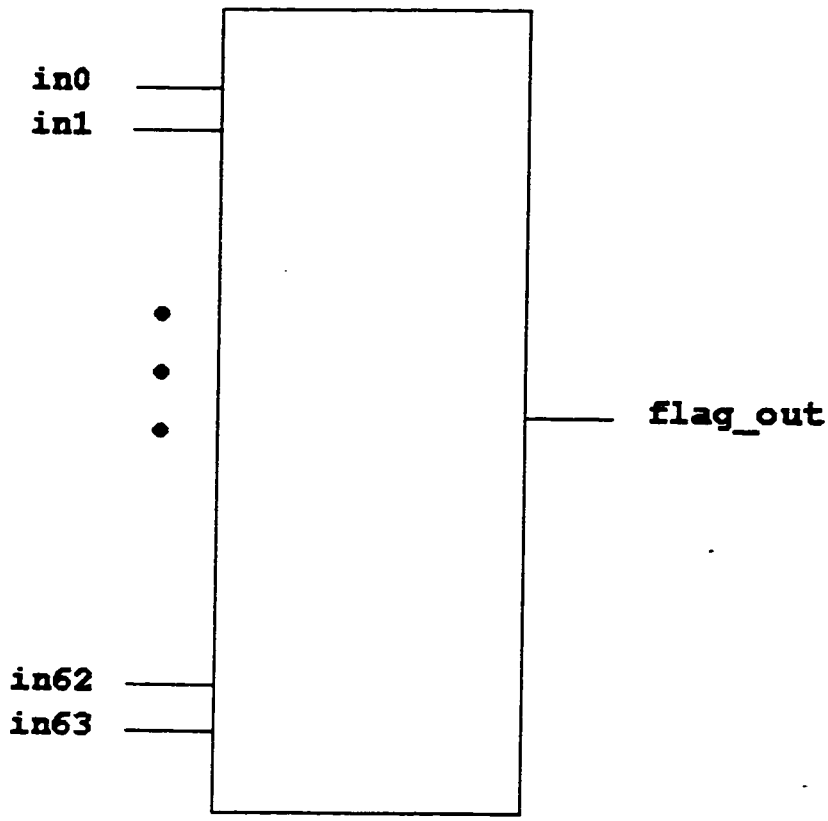


Figure 4-16 : preamble identifier component

The preamble identifier component constantly checks its input against the pre-defined Ethernet preamble. When a match is found, it sets its output to " 0" . Exclusive OR is the basic logic used here. Due to the large number of bits to check and the technology of the Flex 10K device, the processing delay if quite significant. The delay is around 40 ns. This puts a limit as to how fast the system clock can run. Since, once an Ethernet preamble is found, the shift control component has to be able to stop the clock for the data shift component, which is the system clock, within one clock cycle.

4.6.4 Input Control Component

The data shift component shifts in data bits from the input FIFO buffer. However, due to the high speed system clock, which is around 20 to 25 MHz, relative to the write clock, which is usually several megahertz, of the input FIFO buffer, the data shift component may read out the content of the input FIFO buffer faster than it is written in, hence emptying the input FIFO buffer before the next bit is received by the input FIFO. This will result in constant " bubble" inside the data shift component. To avoid this problem, the input control component is introduced. The diagram of the input control component is shown in Figure 4-17. The component has four inputs and three outputs. The data input is connected to the data output of the input FIFO buffer. The output ready signal input is connected to the OR (out ready) output of the input FIFO. The half_full input is connected to the HF (half full) output of the input FIFO. The clock_in input is connected to the clk_2 output of the shift control component. The clock_out output sends the clock signal to the input FIFO as the read clock. The data_out output and the output ready signal (or_out) output are connected to the input of the data shift component and the output ready signal shift component, respectively.

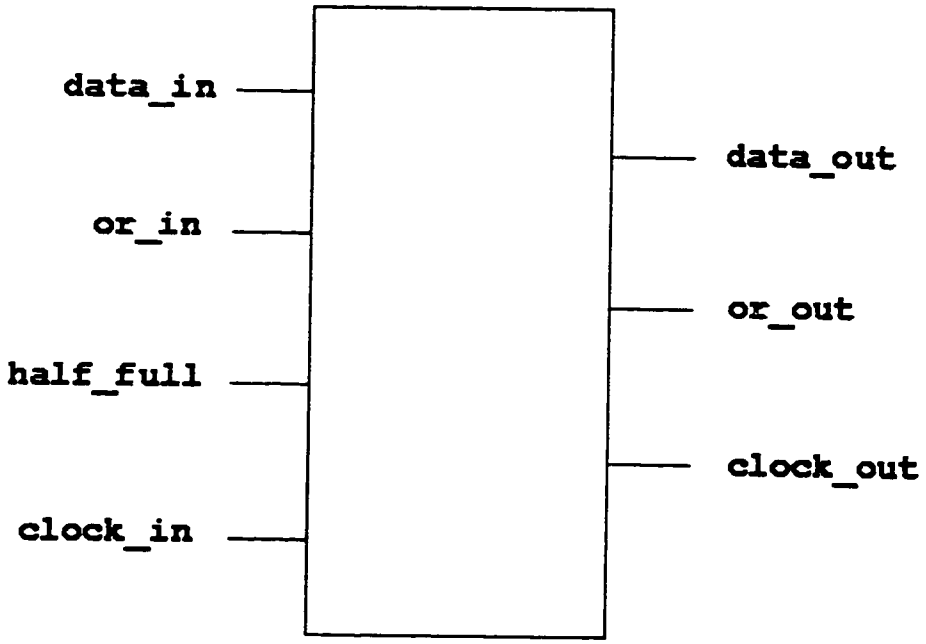


Figure 4-17 : input control component

The input control component monitors the OR port and the HF port of the input FIFO buffer. If the output ready signal is " 0" , then " 0" is sent to the data output. If the output ready signal is " 1" , the input control component checks the half_full input. If the input FIFO has not become half full yet after it starts being filled with data, the input control component holds the output clock to the input FIFO. This avoids shifting data out of the input FIFO before it becomes half full. The FIFO used in this design has a size of 256 bits. When the input FIFO is half full, the output clock is enabled and data bits are shifted from the input into the data shift component. The input control component keeps on passing data bits and the output ready signal from the input FIFO buffer to transmit module until the output ready output of the input FIFO becomes " 0" again. Then the input wait for the next half full signal from the input FIFO to enable the data passing again.

The implementation of the input control component is effective to get rid of the " bubble" among real data bits. It helps to simplify the processing needs of the shift control component.

4.6.5 Preclear Signal Generation Component

The design of the preclear signal generation component is the same as the one described in section 4.2.1. There are two preclear signal generation components in transmit control module. One has both its I and I_EN inputs connected to the output of the output ready signal shift component, while the other one has its I input connected to the send_empty output of the communications control component and its I_EN input connected to the clear_2_1_enable output on the send signal control component in the station control module. The outputs of these two preclear signal generation components are connected to the PC port of the five-bit counter through an OR gate.

There are two situations where the transmitting process needs to be initiated. The most common one is when the send_empty signal is received from the send signal control component in the station control module. In this case, the corresponding preclear signal generation component sends out the preclear signal to the counter and the counter starts counting and provides the clock signal to the transmit module. However, the generation of the final send signal requires that an Ack packet has to be received by the station. There are two reasons to start loading data as soon as the send_empty signal is received. One is, since last transmission is successful, there is no need to retain the old data any more. The other one is, by starting early, the loading process will not put any time constraints on the duration of bus clock cycle. Since, if we wait until the empty packet comes in, then start loading, the bus clock cycle needs to be long enough to let the data loading process finishes before the next clock cycle.

The second situation arises when the above requirement can not be fulfilled. This happens when the first Ethernet packet is shifted in from the input FIFO buffer after the system start up, or, another Ethernet packet arrives after the input FIFO buffer has been

emptied up. In this case, there is no immediate preceding transmission from this station, hence, no Ack packet to be received. New mechanism needs to be established to tell the counter to clear to zero and start counting. Another preclear signal generation component is used for this purpose. The input of this preclear signal generation component is connected to the output of the output ready signal shift component. When the first Ethernet packet comes in, the output ready signal changes from " 0" to " 1" . The preclear signal generation component senses this change and sends out the clear signal to the counter. In the meantime, the clear_2_1_enable signal is set to " 0" , which disables the first preclear signal generator component to prevent any clear signal being generated from it. The outputs of both preclear signal generation components are connected to the inputs of an OR gate. The output of the OR gate is then connected to the PC input of the counter. Therefore, either one of the situations can trigger the transmitting process.

4.7 MAC Address Conversion Module

When a new Ethernet packet is shifted in, the station needs to find out which station this Ethernet packet should be sent to. In order to accomplish this, the MAC address conversion module is used to extract the MAC address of the destination DTE from the header of the Ethernet packet and convert the MAC address to corresponding station address. The module consists of a MAC address register, a three-bit counter, a preclear signal generation component, a static RAM and a MAC address converter.

4.7.1 MAC Address Register

The MAC address register is used to latch the 48 bit destination MAC address following the Ethernet preamble. A diagram of the MAC address register is shown in Figure 4-18. The register has 48 data inputs, a clock input and 48 data outputs. These data inputs are connected to output 63 through output 110 of the data shift component in the transmit module. The outputs are connected to the MAC address converter. The clock input is connected to the output of the preamble identifier component in the transmit control module.

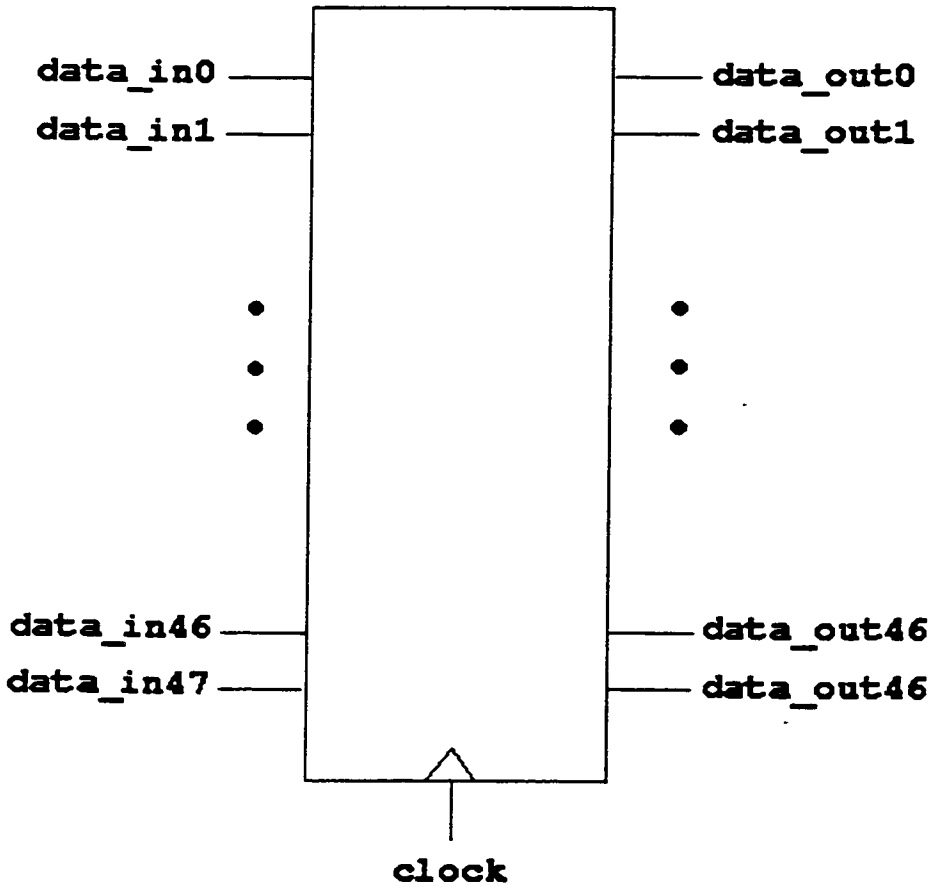


Figure 4-18 : MAC address register

When the Ethernet preamble is found, the output of the preamble identifier component becomes " 0" . At the beginning of the next transmitting process, the output changes to " 1" as the first bit of this Ethernet packet being shifted out. The output is used as a clock signal for the MAC address register. At this time, the 48 bits of MAC address are held at register 63 to register 110. This rising edge of the clock signal latches the 48 bits MAC address into the MAC address register. The MAC address is held at the MAC address register until the next Ethernet packet comes in.

The MAC address is latched for destination MAC address to station address conversion. The MAC address register can be triggered at the falling edge of the clock signal. However, the concern here is that, at the falling edge of the clock signal, the new Ethernet packet has just arrived, e.g. the Ethernet preamble has just been detected. At this moment, the previous transmitting action may not have finished, or retransmission might even be required. Therefore, the previous result from the address conversion needs to be maintained. It is safe to latch in MAC address at the rising edge of the clock as the previous transmission is successfully finished and the new transmission begins.

4.7.2 Three-bit Counter

A three-bit counter is used to control the address conversion process. This LAN on a chip implementation is designed to have seven stations, station 0 to station 6, on the ring. The counter counts from " 000" to " 110" . A diagram of the three-bit counter is shown in Figure 4-19. The counter has a powerup_reset input, a clock input, a clear input and a count_out output. The powerup_reset signal set the counter to " 110" at system start up. The clock input takes in the address_convert_clock as the clock signal for the counter. The clear input is connected to the output of the preamble identifier component in the transmit control module. The count_out output is connected to static RAM and the MAC address converter.

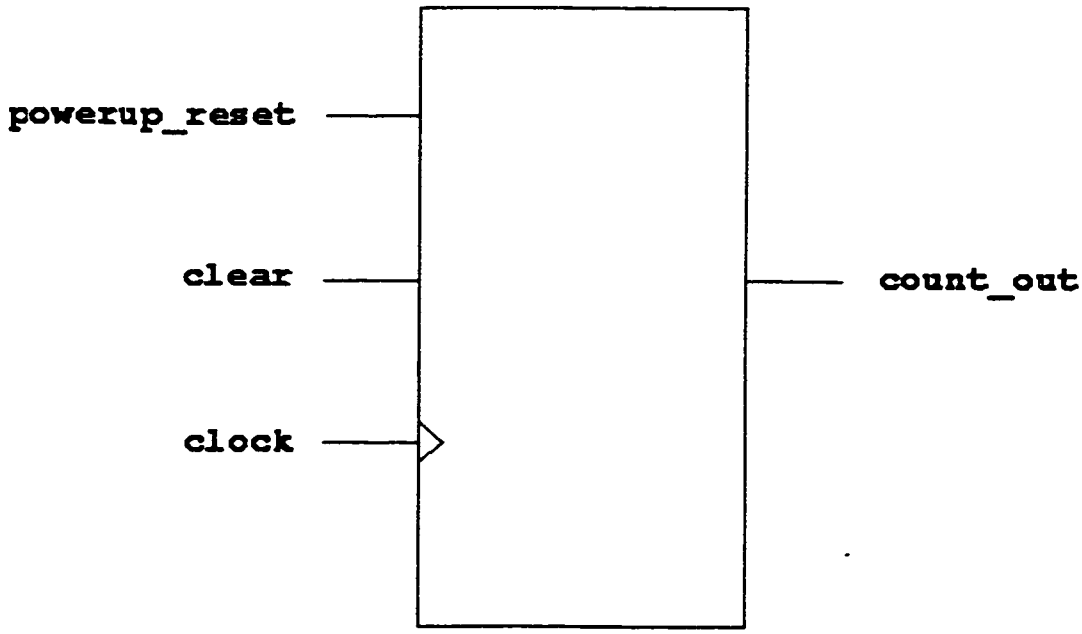


Figure 4-19 : three-bit counter

The method for designing five-bit counter is used for the three-bit counter design as well. The three-bit counter is designed as a finite state machine with eight states. Each state has a corresponding count output which is assigned a binary value, from " 000" to " 111" . The assignment is shown in Table 4-2.

STATE #	COUNT OUTPUT
state 0	111
state 1	101
state 2	100
state 3	000
state 4	001
state 5	011
state 6	010
state 7	110

Table 4-2 : state assignment for three-bit counter

At system start up, the count is set to state 7 and a count output of " 110" is produced. When the counter receives a clear signal, the counter goes to state 0 and the count output changes to " 111" . This starts the counting process. At the rising edge of every address_convert_clock, the counter changes from one state to the next. The count output changes accordingly. The count outputs are used as station address inputs for the static RAM and the MAC address converter. There is only one bit changing logic level from one state to the next one. This eliminates the transient count output and gives the static RAM and the MAC address converter a stable change of station address input each time. The counting process ends at state 7. The counter stays at state 7 until the next clear signal is received.

4.7.3 Preclear Signal Generation Component

The preclear signal generation component is used here, again, to generate clear signal for the three-bit counter. The clear signal starts the counting process. The input of the preclear signal generation component is connected to the preamble identifier component inside the transmit control module. The output is connected to the clear input of the three-bit counter.

When the output of the preamble identifier component changes from " 0" to " 1" , the preclear signal generation component generate a clear signal, which is a " 1" with of duration of one preclear_generation_clock clock cycle. The clear signal sets the count to state 7 and the count output to " 110" .

4.7.4 Static RAM

A small static RAM is created to store the MAC addresses and station addresses used in the address conversion. The diagram of the static RAM is shown in Figure 4-20. The static RAM has two enable inputs, we and oe. These two enable pins are connected to external input signals. The write_stn_addr and mac_addr_input input buses are also connected to external signals. These four inputs are used during system set up to load the

static RAM with the MAC address of each station on the ring. The static RAM also has one read_stn_addr input bus and a mac_addr_output output bus. The read_stn_addr input is connected to the count output port of the three-bit counter. The mac_addr_output output is connected to the MAC address converter.

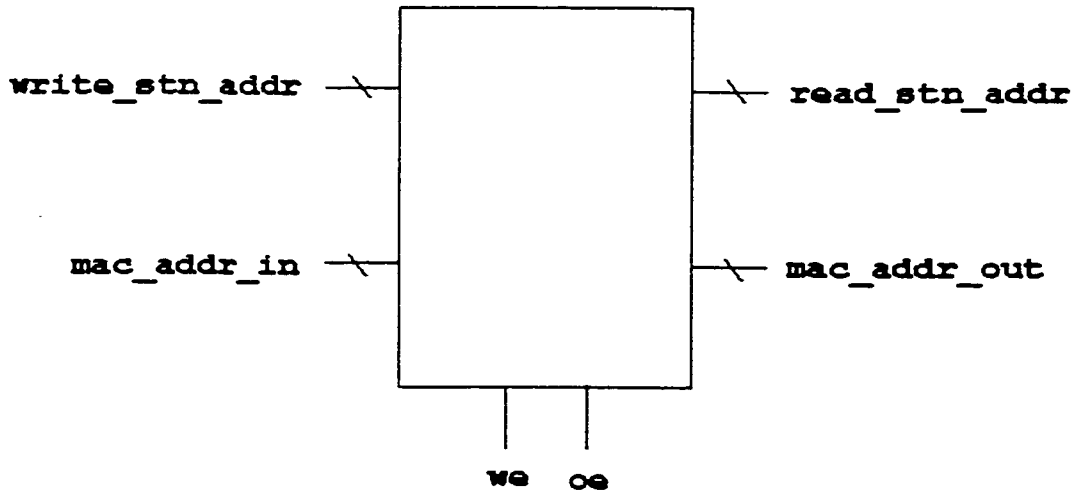


Figure 4-20 : static RAM

During the system set up, the *we* input is set to " 0" and the *oe* input is set to " 1" . This enables the writing of the MAC address of each station into the RAM. The writing of the static RAM contents is not automated in this implementation. To achieve plug-and-play, extra hardware and software are needed to automatically recognize the MAC address of each DTE attached to the ring. Each station address and MAC address pair is presented to the RAM with the station address as the addresses of the RAM and the MAC address as the contents. After every MAC addresses have been loaded into the RAM, the *we* input is set to " 1" and the *oe* input is set to " 0" . This enables the read operation on the static RAM during the address conversion.

During the MAC address conversion process, the three-bit counter generates the station address as its count output and sends these station addresses to the static RAM, one at a

time, through the read_stn_addr input bus. As a result, each MAC address is presented at the mac_addr_output, which sends the MAC address to the MAC address converter.

4.7.5 MAC Address Converter

The diagram of the MAC address converter is shown in Figure 4-21. The MAC address converter is used to determine the corresponding destination station address of the Ethernet packet to be transmitted. The MAC address converter has four inputs and one output. The convert_enable input, which is connected to the address_convert_clock. The mac_addr_in is connected to the output of the MAC address register. The ram_mac_addr_in is connected to the mac_addr_output of the static RAM. The stn_addr_in input is connected to the count output of the three-bit counter. The stn_addr_out output is connected to the output module where the out going packets are assembled.

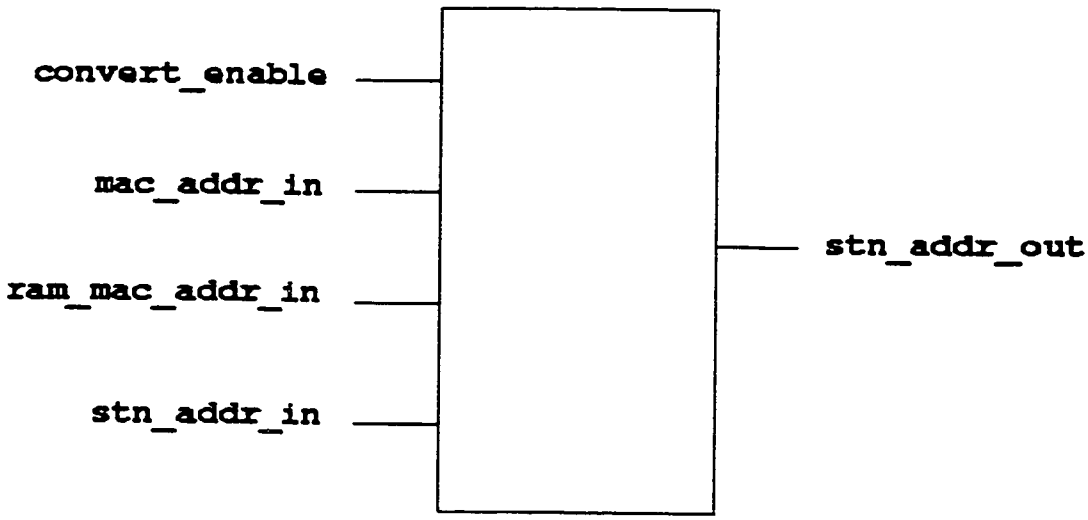


Figure 4-21 : MAC address converter

When a new Ethernet packet arrives and is shifted into the series-to-parallel converter, the output of the preamble identifier component changes from " 0" to " 1" . This latches

the destination MAC address into the MAC address register. In the meantime, this also causes the preclear signal generation component sends a clear signal to the three-bit counter which starts the counting process. As the counter counts, each station address is sent to the static RAM and the MAC address converter, one by one. The static RAM sends the corresponding MAC addresses to the MAC address converter. The MAC address converter compares the MAC address from the MAC address register with the MAC address from the static RAM. Once a match is found, the station address at the MAC address converter' s input is latched and presented at its output bus. The latched station address stays on the stn_addr_out output bus until the next address conversion process determines a new station address.

The processing delay on the static RAM is quite significant and the output MAC address is not very stable when the input station address changes. Therefore, the convert_enable input is used to shield the MAC address converter from these unstable MAC address output from the static RAM. The MAC address converter is enabled when the convert_enable is " 0" . As described above, the convert_enable input takes in the address_convert_clock. The rising edge of this clock causes the three-bit counter to send out new station address and, consequently, causes the static RAM to send new MAC address. The clock signal has a duty cycle of 50%. When the falling edge arrives, i.e. the convert_enable becomes " 0" , the MAC address from the static RAM has been stable already. This prevents the MAC address converter from latching an incorrect station address when the MAC address from the static RAM is still unstable.

4.8 Output Module

The output module is where the outgoing packets are assembled, under the control from the station control module. There is only one component, output control component, in the output module.

4.8.1 Output Control Component

The output control component is basically a multiplexer. A diagram of the component is shown in Figure 4-22. It has five control inputs, send_ack, send_nack, send_empty, send_data and resend. They are connected to the corresponding outputs signals in the station control module. The latch_in input bus is connected to the output of the station latch. The shift_data_in input bus is connected to the outputs of the series-to-parallel converter. The shift_dest_in input is connected to the output of the MAC address converter. The shift_flg_2 is connected to the output of an AND gate which has the outputs from the preamble identifier component and the output ready signal shift component as its input. The shift_flg_2 provides the group indicator signal for the outgoing packet. The packet_out output bus is connected to the data bus.

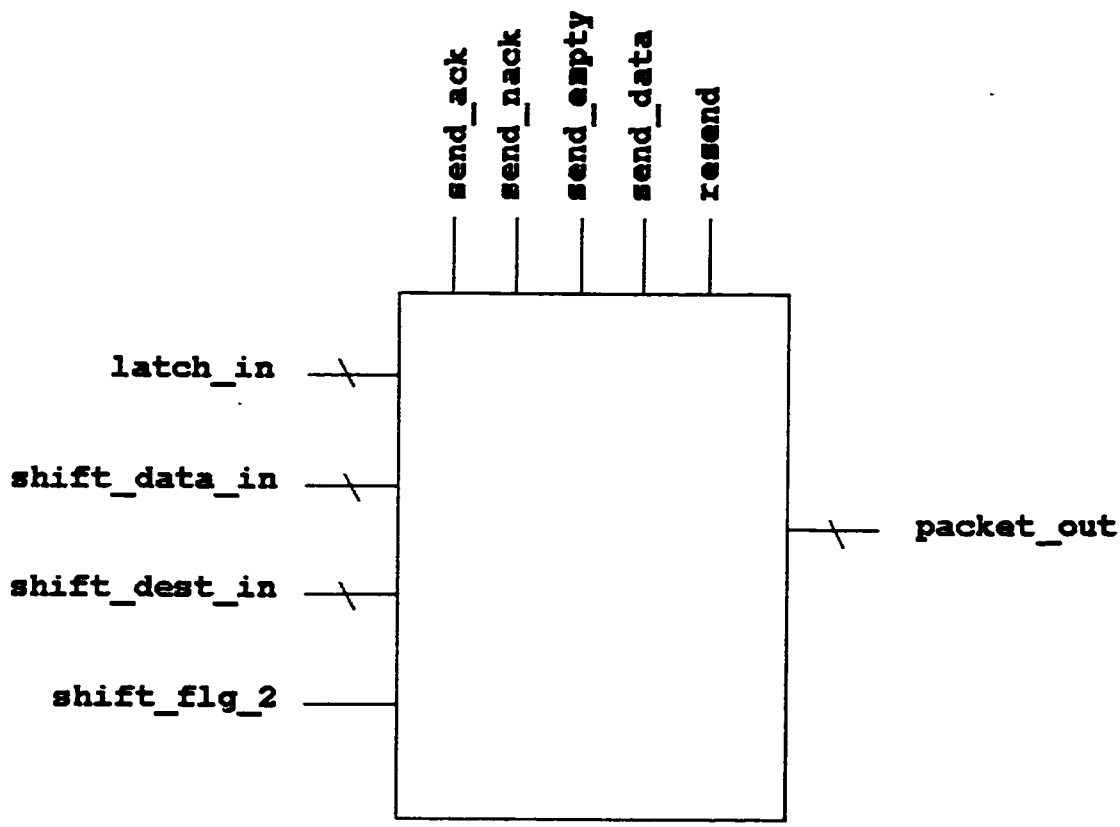


Figure 4-22 : output control component

When send_ack, send_nack or send_empty signal are received, the output control component puts appropriate source station address, destination station address and packet type flag on the output and set the data field and group indicator to " 0" . When the send_data or resend signal are received, the output control component puts this station' s address as the source station address, the output of the MAC address converter as the destination address, sets the output flag to " 11" , puts the output of the series-to-parallel as the output data and the group indicator is taken from the shift_flg_2 input. When the output of the preamble identifier component and the output ready signal are both " 1" , the output group indicator is " 1" . When either that the preamble is found, i.e. the end of the previous Ethernet packet, or there is no data to send, i.e. the output ready signal is low, the group indicator will be " 0" .

When none of the above five control signals is received, the output control component knows that the station control module did not take any action on the packet received, i.e. the incoming packet is not meant for this station. Therefore, the output control component puts the output of the station latch on output bus, hence passes the received packet to the next station.

At the rising edge of the bus clock, the output of the output control component is latched into the station latch of the next station on the ring.

4.9 FIFO

The FIFO buffer used in this implementation was not custom design in VHDL. Instead, the ti2229 256x1 FIFO, from Texas Instrument, is used as external FIFO. There are two FIFO buffers for each station. One is for receiving, the other one is for transmitting. The diagram of the ti2229 component is shown in Figure 4-23. The ti2229 FIFO memory supports clock frequencies up to 60 Mhz [11].

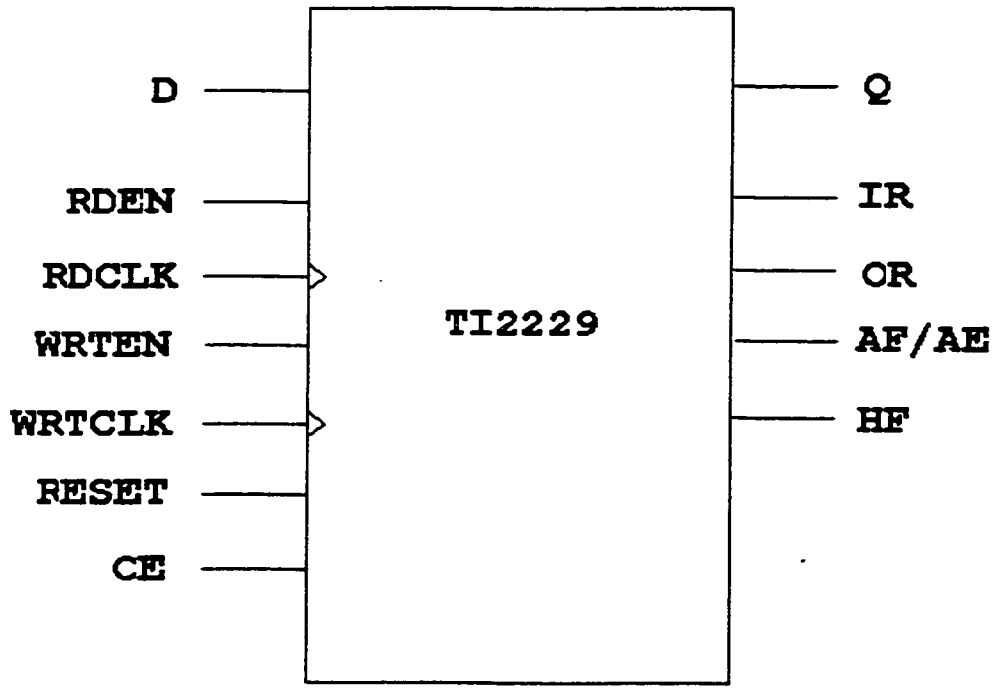


Figure 4-23 : FIFO

For the receiving FIFO buffer, the data input is connected to the output of the parallel-to-series converter, while write clock is controlled by the five-bit counter inside the receive control module. The IR output is connected to the station control module. For the transmitting FIFO buffer, the data output is connected to the input of the input control component inside the transmit control module. The read clock is controlled by the five-bit counter in the transmit control module. The OR and the HF outputs are connected to the output ready signal and half_full inputs of the input control component as well. All other inputs of FIFO buffers are controlled by external signals.

Chapter 5

Analysis of Ethernet Architecture by Computer Simulation

5.1 Experiment Setup

The experiment was conducted by performing network simulation using the OPNet network simulator from MIL 3, Inc.. In this experiment, we used one station as a video server, three stations as video display terminals and an eight port Ethernet hub to connect these stations. The overall configuration is shown in Figure 5-1.

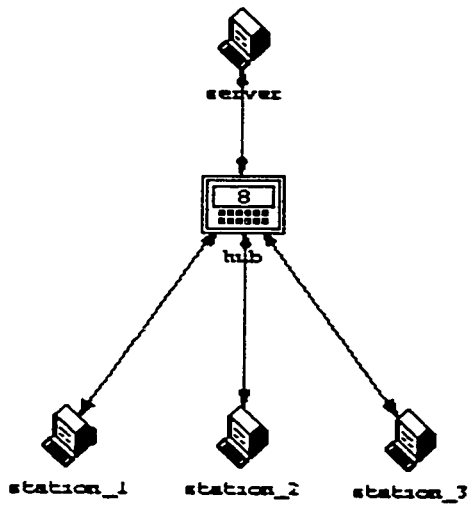


Figure 5-1 : network simulation configuration

The video server generates three MPEG-2 video streams and sends these video streams to video display terminals through the Ethernet hub. Each of the three video streams goes to one of the video terminals, respectively. The connections between stations and the hub are high speed duplex links. In this scenario, the assumption was made that the video streams arrives at the hub with no delay. Therefore, the only delay the packets experience is introduced by the Ethernet data bus inside the hub. This is consistent with what we want to study in this experiment, that is, to observe the time delay introduced by the Ethernet data bus, not the access link.

The server node model, shown in Figure 5-2, has five components. The generator produces packets with MPEG-2 video data as payload. The sink receives and destroys packets containing interactive control data, e.g. pause, fast forward, rewind, etc., sent back by video terminals. The MAC processor implements the medium access control algorithm. The transmitter sends data packets to the hub while the receiver receives data packets from the hub.

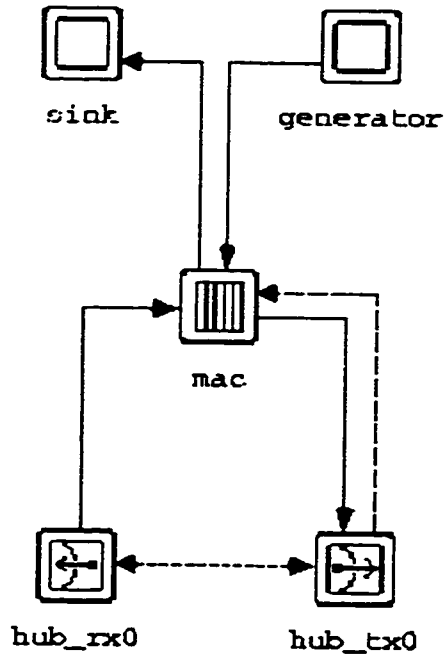


Figure 5-2 : video server model

The video terminals have the same base node model as shown in Figure 5-2. However, the generator in the video terminals does not generate video streams. Instead, signals with much lower bit rate is generated to simulate upstream traffic such as user interactive control signals like pause, rewind, fast forward. The sink receives and destroys video packets.

To simulate the Ethernet data bus, an Ethernet hub was used. Since Ethernet hub is basically an Ethernet data bus inside a box. The Ethernet hub node model is shown in Figure 5-3. The hub process takes the packet received from any input port and broadcasts the packet to every ports. This is exactly what Ethernet data bus does when it receives a packet. An eight port Ethernet hub was used in this experiment. The transmitters and receivers transmit data packets to and receive data packets from the connected device, respectively.

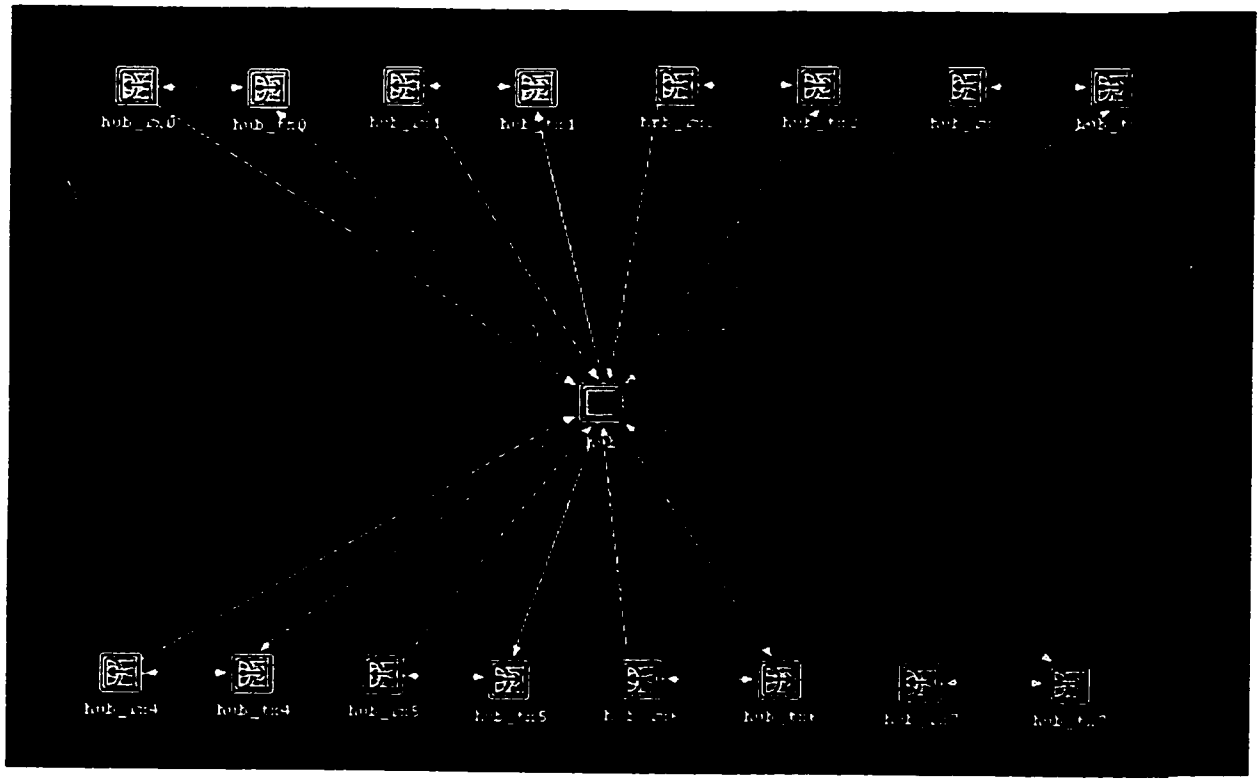


Figure 5-3 : hub model

The link model simulates a duplex point-to-point link which can support 100 Gbps bandwidth requirement.

The reason for chosing high bit rate data transfer in this experiment is to measure the time delay introduced by the Ethernet data bus itself, not the link between the Ethernet bus and DTEs. By setting a high transmission rate on the link, the queuing delay due to busy transmitter was eliminated as much as possible. Hence, the only delay a packet might experience was the delay happened inside the hub, i.e. on the Ethernet bus.

The experiment was conducted with the following parameters and circumstances.

- The MPEG-2 video streams were generated at 9 Mbps, which in turn required the video server to send out data at roughly 3000 packets/sec.
- The user control data packets were sent out every 10 seconds to simulate situation where viewer frequently does fast forward or rewinding.

- The video data was carried by Ethernet packet with 9600 bits in length.
- The user interactive control data was carried by Ethernet packet with 1024 bits in length.
- The sending and receiving of three video clips of 60 seconds long were simulated in this experiment.
- The bit rate supported by the links between the hub and the video server and the hub and the video terminals was 100 Gbps.
- The hub used is a 100Base-T Ethernet hub which support data transmission rate of 100 Mbps.

The short duration of the video clip was chosen because simulation of longer video session requires very lengthy simulation. Another constraint on the duration of the video session simulated was imposed by the memory space available on the SUN machine that runs the simulation.

5.2 Description of MPEG-2

In this experiment, we tried to simulate network behavior with real-time application traffic. The bit rate used in this experiment was derived from MPEG-2 video traffic profile.

MPEG-2 video standard is video coding standard. It is defined by Moving Picture Export Group (MPEG) along with MPEG-2 audio and MPEG-2 system. Its primary application targeted during the MPEG-2 definition process was the all-digital transmission of broadcast TV quality video at coded bit rate between 4 and 9 Mbps. However, the MPEG-2 video standard has been found to be efficient for other applications such as those at higher bit rates and sample rates, e.g. HDTV (High Definition Television).

The MPEG-2 video traffic simulated in the experiment complies with CCIR-601 standard which requires a sampling dimension of 720x480 and sampling rate at 30 frames per

second. The MPEG-2 video stream bit rate ranges from 4 Mbps to 15 Mbps depending on the complexity of spatial-temporal contents and type of video quality needs to be achieved. For studio TV quality, the CCIR-601 standard requires a maximum bit rate at 15 Mbps.

5.3 Simulation Results

The simulations of MPEG-2 video traffic on 10Base-T Ethernet network were conducted under two scenarios. In both cases, the end to end delay of each packet was captured for further analysis.

In the first scenario, the only data stream appeared on the Ethernet data bus was the MPEG-2 video stream coming from the video server. This gave us the time delay pattern of all packets when there were no other parties trying to access the data bus at the same time. Therefore, there were no contentions and collisions.

In the second scenario, the user interactive data streams coming from the three video display terminals were added to the traffic. This caused collisions on the Ethernet data bus. Hence, we should be able to observe the change in time delay pattern.

Table 5-1 lists the simulations that have been conducted on the Ethernet network setup described in Section 5.1. The downstream traffic is the video stream coming from the video server. The upstream traffic is the user interaction data stream coming from the video display terminals.

transmission rate	downstream packet size (bits/packet)	upstream packet size (bits/packet)	MPEG-2 bit rate	server generation rate (packets/sec)	without user interaction	with user interaction
100 Gbps	9600	1024	9 Mbps	3000	eth_1	eth_2

Table 5-1 : parameters setup in network simulation

Now, we can take a look at the simulation results. Figure 5-4 shows the packet end to end delay from simulation eth_1. On the bottom of this figure, there is a solid line which is very close to zero. If we zoom in, we can find out that this solid line represents the minimum delay a large number of video packets experienced. This minimum delay, which is about 0.2μs (microsecond), occurs when there is no collision on the Ethernet data bus inside the Ethernet hub, no queuing delay at the transmitter and the video packets only experienced processing delay occurred at the Ethernet hub. Figure 5-5 shows the average delay over time is approximately 0.35μs. This indicates that a great majority of packets experienced minimum delay shown by the solid line in Figure 5-4. The average delay is slightly higher than the minimum due to longer delays experienced by some packet appear above the solid line in Figure 5-4.

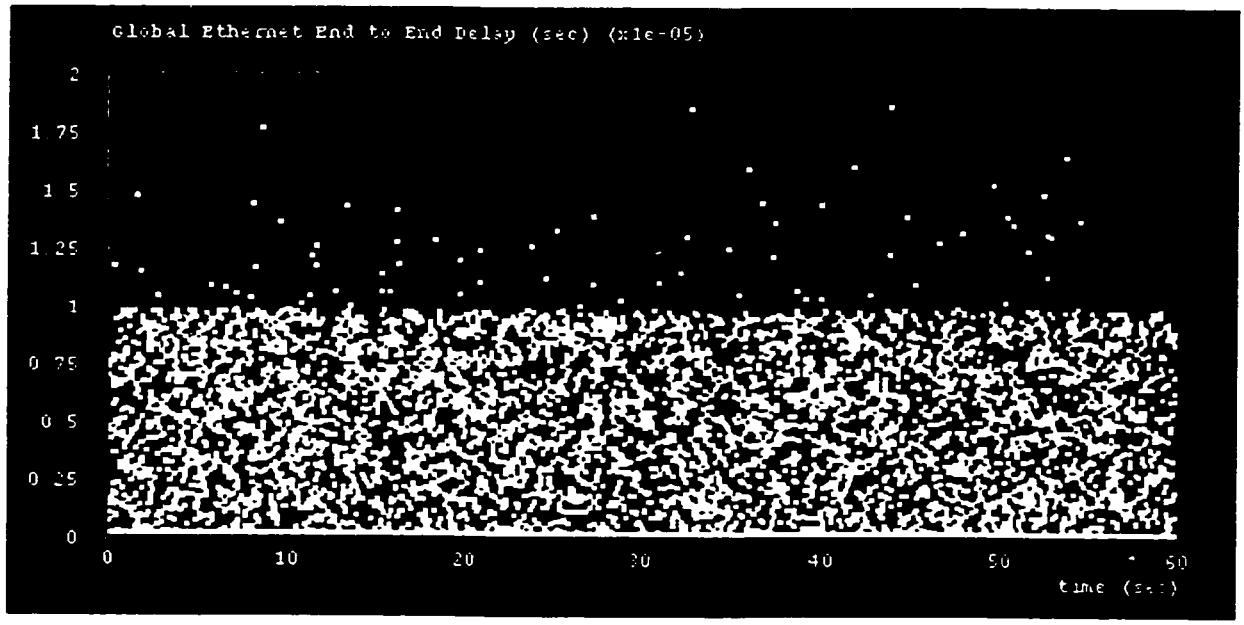


Figure 5-4 : results from simulation

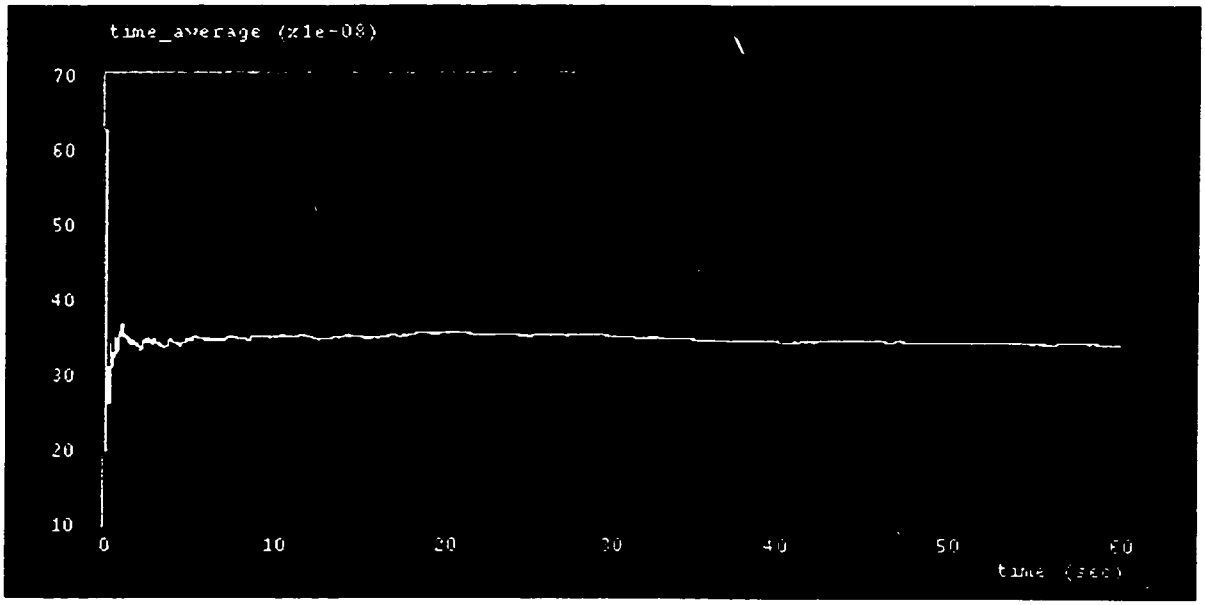


Figure 5-5 : average time delay over time

In Figure 5-4, we can also see a big band of dots that represents packet end to end delay with value up to 10μs. This tells us that a large percentage of video packets experienced queuing delay during transmission. Above 10μs mark, there are some scattered dots which represent those video packets that have been severely delayed. This may occur when the instantaneous generation rate of the video server exceeds the processing rate of the Ethernet hub. Therefore, some queuing delay inside the hub is accumulated. The video server' s video packet generation process follows a Poisson process with a mean of 3000 packets/sec. From Figure 5-4, we can see the maximum end to end delay is about 18.5μs. We also noticed that there were 71 packets suffered severe delay.

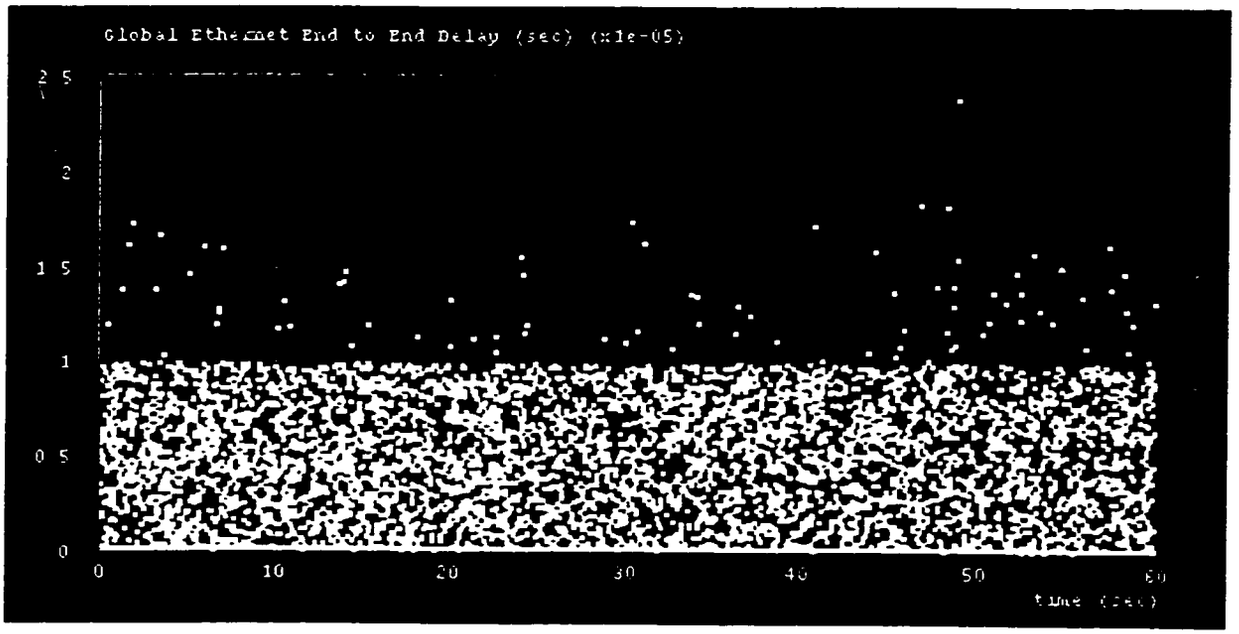


Figure 5-6 : results from simulation

Figure 5-6 shows the end to end delay for the simulation eth_2. In this figure, there are 14 dots under the 0.2μs mark (solid line) can be observed. Some of these dots can be found in between and around 10 second and 20 second mark and also around 55 second mark. These dots represent the delay experienced by the user interactive control data packets. Since these packets are shorter in length (1024 bits/packet), the processing time is also shorter. The pattern presented in Figure 5-6 is very similar to that in Figure 5-4. There are 85 dots that are above the 10μs mark, Comparing with Figure 5-4, 14 more dots are recorded. This shows that an extra 14 video packets suffered severe time delay. Since no other conditions have been changed during the simulation, it can be concluded that the collision between the video data packets and the user interactive control data packets caused the extra video packets being severely delayed.

This simulation gives us the maximum end to end delay of approximately 24μs, which is a 29.73% increase from 18.5μs in the previous simulation. We also noticed that, in Figure 5-4, there are only 7 packets experienced time delay ranging from 15μs to 20μs. However, in Figure 5-6, there are 16 packets that have time delay falls in this range, a roughly 130% increase. This indicates that collision between packets on the shared

Ethernet data bus can cause substantial time delay. This delay is the result of random waiting time before the retransmission of the same packet that experienced collision. Comparing to the normal processing delay and transmission delay, this random delay may vary in a quite large range causing jitter in real-time applications.

Real-time applications, such as video, are very sensitive to jitter. The existence of jitter causes a picture to look jerky because not every frame arrive at the same pace. Ethernet architecture is bound to introduce jitter since the random waiting time after collision is not under control. The above observation tells us that Ethernet architecture has some inherent defect when handling real-time applications.

5.4 Summaries

In this section, the impact of Ethernet data bus architecture on real-time applications has been investigated. Two simulations have been conducted with MPEG-2 video as the real-time application traffic. In one scenario, no user interactive control was included. The other scenario included frequent user interactive control data stream in the simulation. These upstream traffic can also represent non-real-time applications such as email, ftp, etc..

The results are studied and lead us to the following conclusion. The Ethernet shared data bus combined with the CSMA/CD MAC protocol gives rise to jitter, an unwanted characteristic in real-time applications, since the waiting time after collision on the Ethernet data bus is random, not well controlled. The random waiting time could be large relative to normal delay when there is no collision. To overcome this problem, new LAN architecture needs to be explored.

Chapter 6

VHDL Simulation of LAN on A Chip

The LAN on a chip architecture uses the P-Bus to pass Ethernet packets to destination stations. To prove the concept, the preliminary performance evaluation and analysis was undertaken by simulating the design with full timing. The design was implemented in VHDL and simulated with ModelTech' s VHDL simulator. First, each individual component was simulated, functionally, to verify its behavior. Then the station was assembled and simulated functionally and with timing as well. Finally, the entire ring with three stations was simulated with timing analysis.

6.1 Tools

There are several major steps to follow in order to simulate the design. Number of tools were used in each steps. First, the entire design was written in VHDL by using general purpose text editor. Secondly, the design was synthesized with Leonardo, a VHDL/Verilog synthesizer from Examplar, to obtain the VHDL netlist. Then, the VHDL netlist was taken as the input to MAXPlus II, the Altera' s place and route tool, to create

the EDIF netlist with Altera' s EPF10K503 as its target device. Finally, the design was simulated with its testbench in ModelSim, the VHDL/Verilog simulator from Model Technology Inc.

6.2 The Simulation Scheme

To simulate the LAN on a chip architecture, the FPGA design implemented a ring with three stations. In the implementation, station 0 is used as an interface between the LAN and the access network. High speed data stream is sent into the input FIFO buffer of station 0. Attempt was made to have a high speed data stream with a bit rate similar to that of the MPEG-2 videos. The purpose was to simulate a situation where two video streams come in from station 0. Different video stream was sent to different station. There video streams were downloaded by station 1 and station 2. In a real situation, station 1 and 2 would be connected to DTEs where videos are played back. The overall configuration is shown in Figure 6-1.

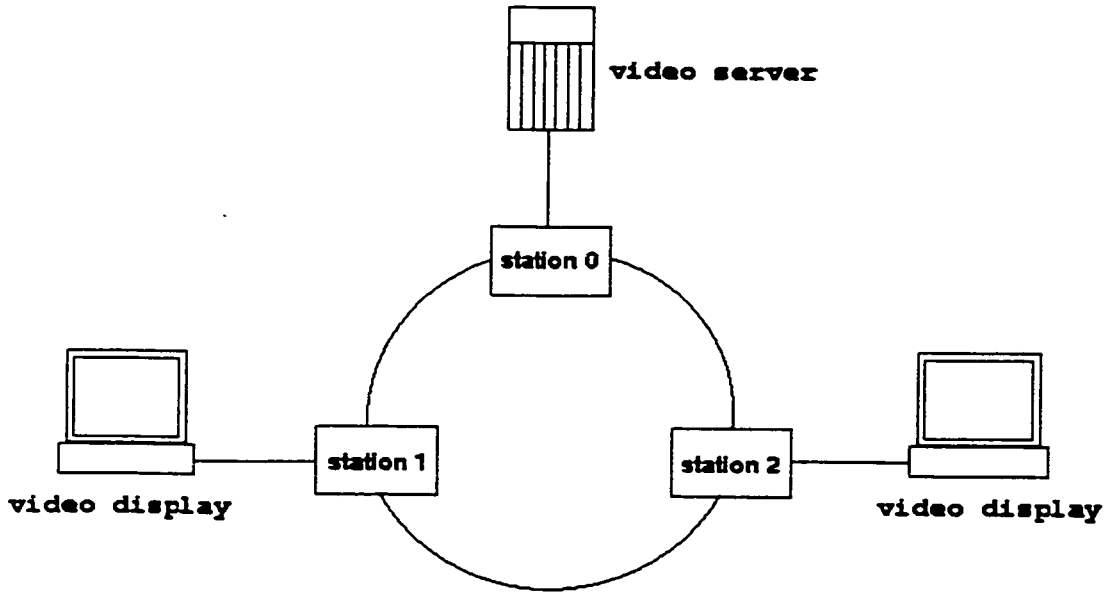


Figure 6-1 : VHDL simulation setup

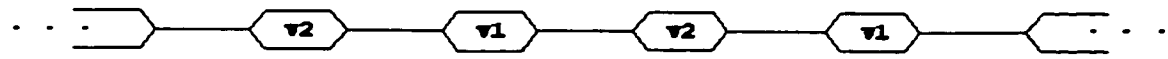
The LAN on a chip design was implemented on the EPF10k100GC503-3 FPGA from Altera. The FIFO memory is ti2229 from Texas Instrument. The testbench, written in VHDL, is responsible for generating all the stimulus needed for simulation. It provides all three input data streams for three stations. It also provides all clock signals to the FPGA device and control signals for FIFO buffers. The specification for these signals are listed in Table 6-1.

bus clock	1 MHz
system clock	20 MHz
system receive clock	33.3 MHz
preclear generation clock	50 MHz
address convert clock	12.5 MHz
ram_oe	0
ram_we	1
fifo_oe	1
fifo_rden	1
fifo_wrten	1
data input 0	10 Mbps
write clock 0	10 MHz
read clock 0	5 MHz
data input 1	5 Mbps
write clock 1	5 MHz
read clock 1	5 MHz
data input 2	5 Mbps
write clock 2	5 MHz
read clock 2	5 MHz
data input 3	5 Mbps
write clock 3	5 MHz
read clock 3	5 MHz

Table 6-1 : parameters used in VHDL simulation

The size of Ethernet packets are 256 bits long. Due to the processing speed of the SUN workstation, the simulation runs slow. It takes about 30 minutes to simulate 3 milliseconds of data transmission on the LAN.

Three sets of simulation were performed in order to compare time delay and jitter. In the first scenario, station 0 sends one data stream to station 1 and another one to station 2. The overall input data stream has the pattern shown in Figure 6-2. The effective data rate for each data stream is 2 Mbps.



v1 : video data stream going to station 1
v2 : video data stream going to station 2

Figure 6-2 : video data stream pattern

This scenario simulates the situation where there is only downstream traffic, for example, downloading video program. The time delays obtained from this simulation are the result of the queuing and processing delays.

In the second scenario, the station 0 still sends two data stream to station 1 and station 2, as it does in the first scenario. The profile of these two data stream are the same as in scenario 1. The only difference is that station 1 sends a data stream back to station 0. This simulates the situation where user interactive control signal is sent back to the video server on the network. For simulation purpose, the control data stream was actually sent at 4 Mbps. The intent was to create more traffic on the ring and find out how this situation affect the downstream video data traffic.

In the third scenario, the same two video data streams are sent out to station 1 and station 2 from station 0. But in this scenario, in addition to the control data stream from station 1, a data stream is also sent from station 2 to station 1. This simulates the data traffic of another application between station 1 and station 2. Both the control data stream and the

data stream from station 2 to station 1 were sent at 2.5 Mbps. In this scenario, collision may occur and video data packets may experience delay due to collision besides processing and queuing delay.

6.3 Simulation Results and Comparison

Time delay occurred to video data packets was measured between the input of the input FIFO of station 0 and the output of the output FIFO of the other two stations.

In scenario 1, the simulation was run for 3 milliseconds worth of data traffic on the LAN and the time delays is recorded in Figure 6-3 and Figure 6-4.

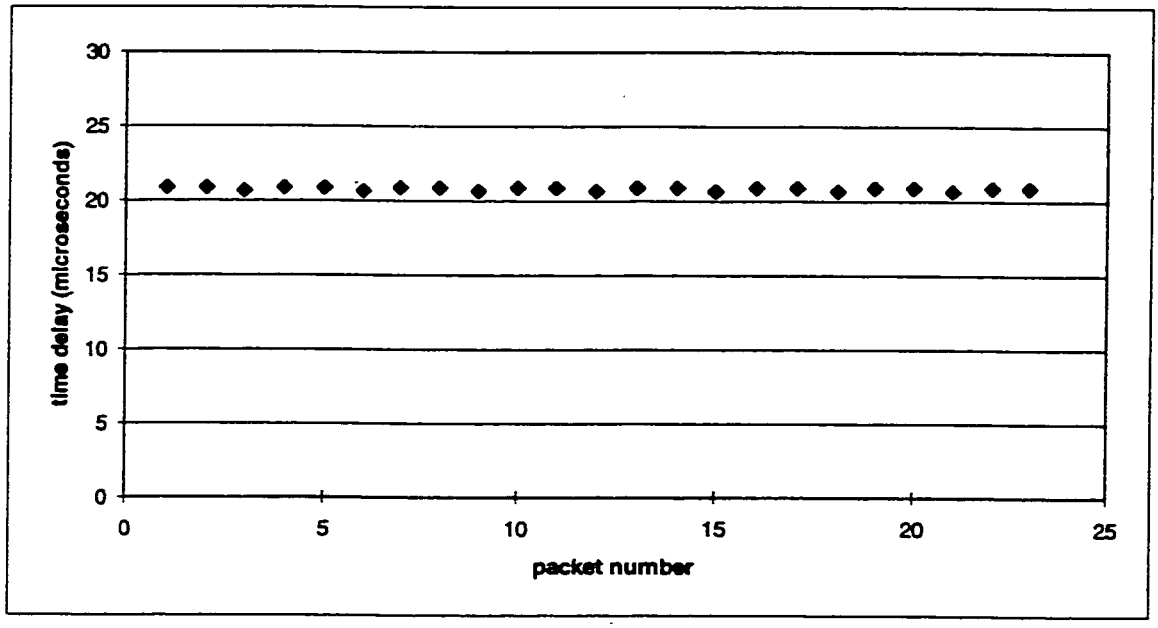


Figure 6-3 : time delay from station 0 to station 1 in scenario 1

Figure 6-3 shows the time delay for video data packets going to station 1. The results show that the time delays are the same with the value of 20868 ns.

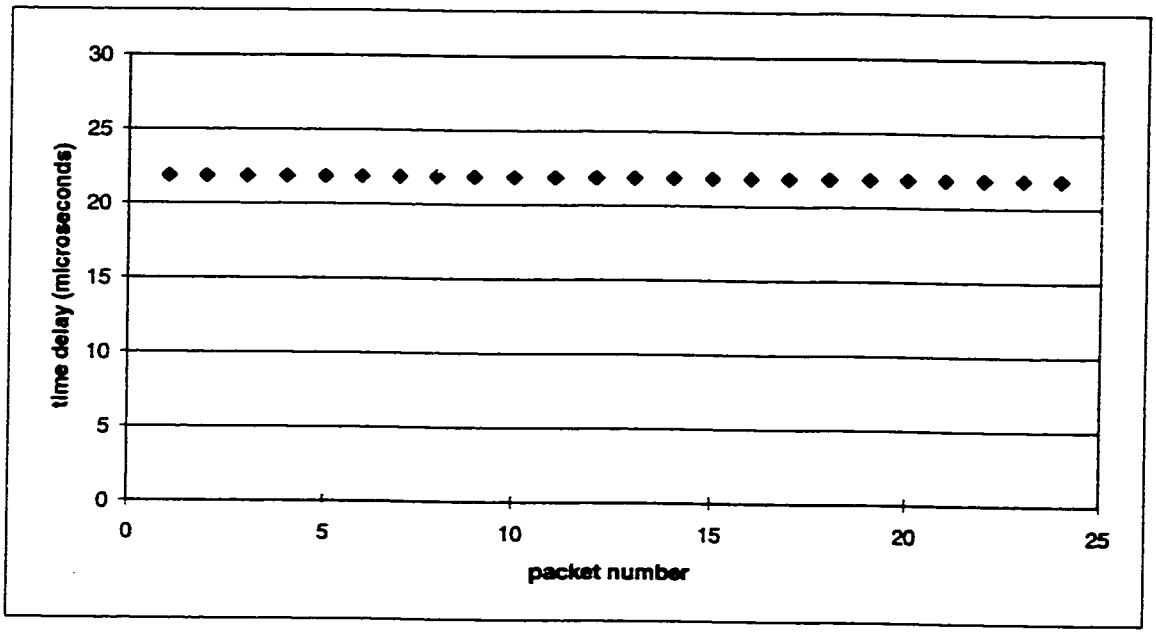


Figure 6-4 : time delay from station 0 to station 2 in scenario 1

Figure 6-4 shows the time delay of video data stream going to station 2. The results also show that the delays take a uniform value of 21868 ns. The extra 1000 ns is due to the fact that video data packets take one more bus clock cycle to reach station 2 than to station 1. Since the bus clock cycle is 1000 ns, therefore the delay between station 0 and station 2 has an extra 1000 ns. In scenario 1, there were no other traffic on the LAN. These time delays are the minimum from station 0 to station 1 and station 2.

In scenario 2, the simulation was also run for 3 milliseconds worth of data transfer. The two video data streams were each sent at 2 Mbps. The control data stream, from station 1 to station 0, was sent at 4 Mbps for test purpose. The time delays were recorded in Figure 6-5 and Figure 6-6.

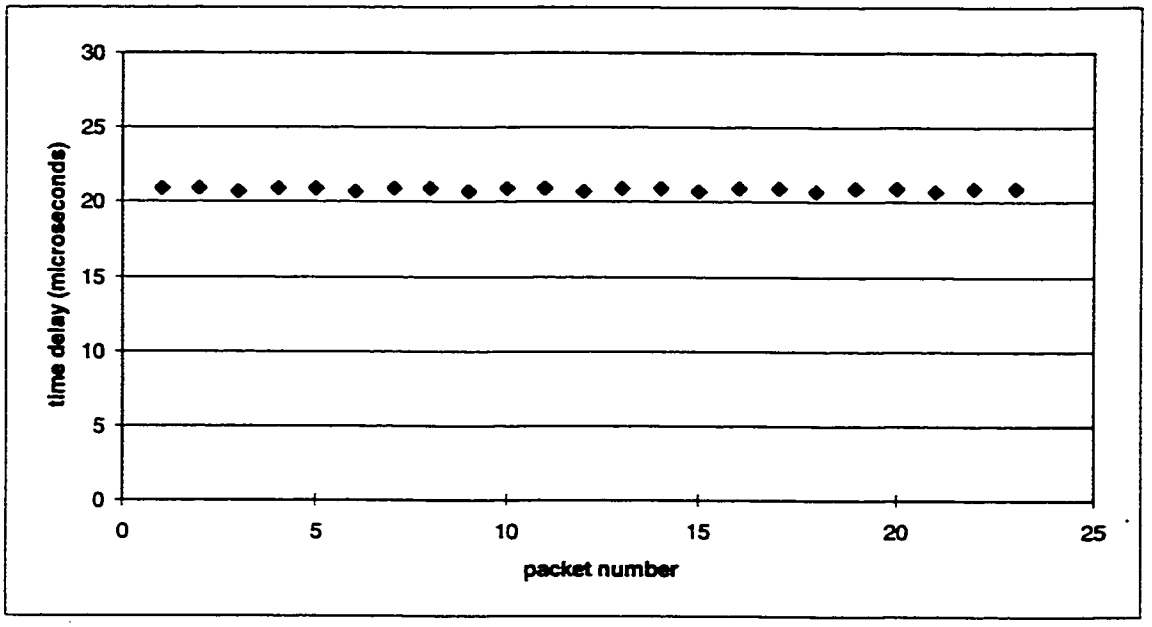


Figure 6-5 : time delay from station 0 to station 1 in scenario 2

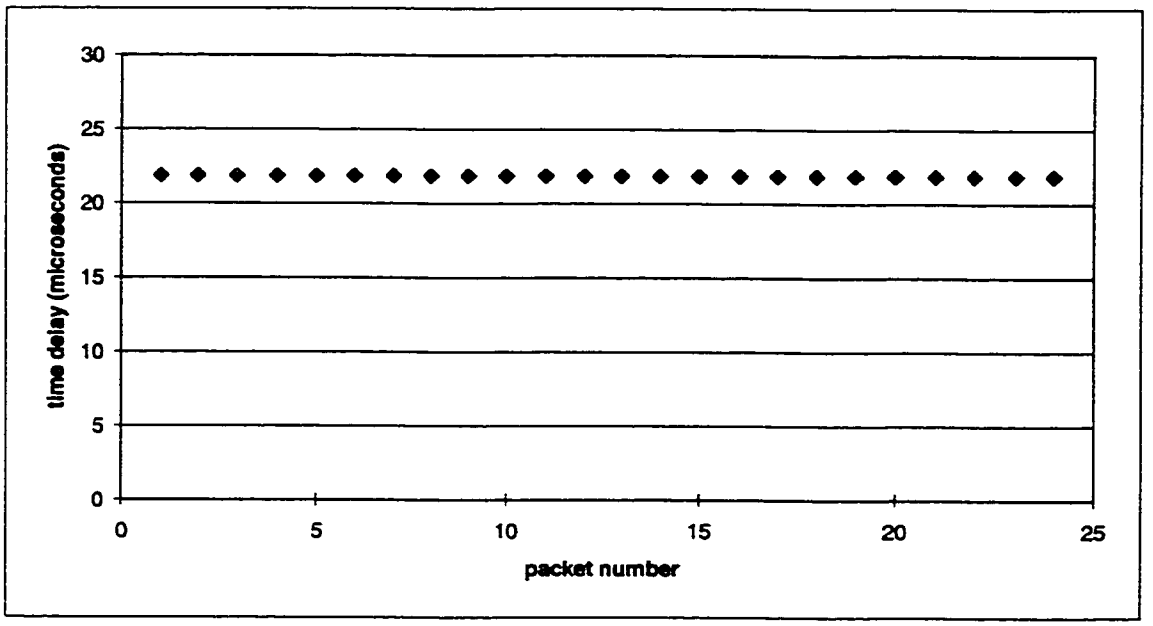


Figure 6-6 : time delay from station 0 to station 2 in scenario 2

Figure 6-5 shows the time delay occurred to video data packets going to station 1 while Figure 6-6 shows time delay for video data packets going to station 2. From the results, we can see that the both sets of delays did not change. That is to say that extra data traffic

on the ring did not affect the video data traffic at all. This shows a fundamental difference between this proposed LAN architecture and the conventional Ethernet common data bus architecture. In the case of the Ethernet shared bus architecture, simultaneous data transmissions of the above sort inevitably result in collisions and, hence, prolonged time delay. However, the proposed LAN architecture implemented in this thesis allows certain type of multiple transmissions occur at the same time without introducing extra delay.

We noticed that, in scenario 2, video data streams and the control data stream went to different destination stations. This type of simultaneous access to the ring does not cause collision, as shown in scenario 2. In scenario 3, we considered another type of simultaneous access to the ring where two data streams have the same destination station. In this situation, collisions are expected since the LAN on a chip architecture was designed in such way that each station is supposed to receive the entire Ethernet data packet before it can receive another one. Therefore whichever Ethernet data packet arrives first at the destination station has the right to finish the transmission while the data packets from the other data stream waiting.

In scenario 3, the simulation was run for 3 milliseconds worth of data transfer as well. The time delays were recorded in Figure 6-7 and Figure 6-8. The video data streams were sent at 2 Mbps each. The control data stream, from station 1 to station 0, was sent at 2.5 Mbps while the other data stream, from station 2 to station 1, was sent at 2.5 Mbps as well.

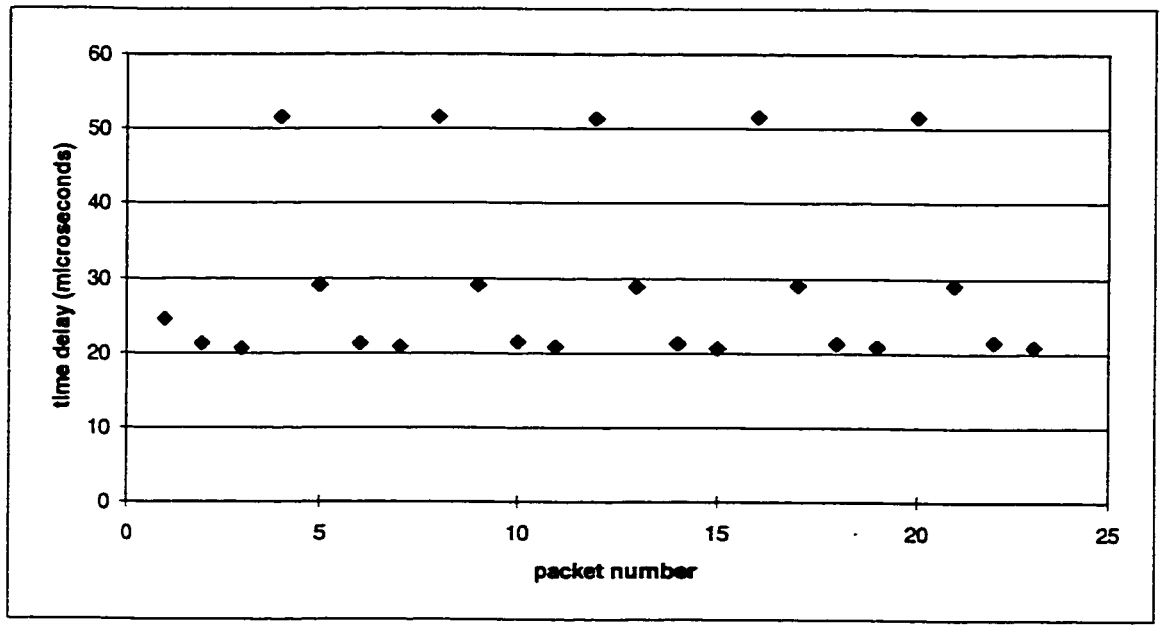


Figure 6-7 : time delay from station 0 to station 1 in scenario 3

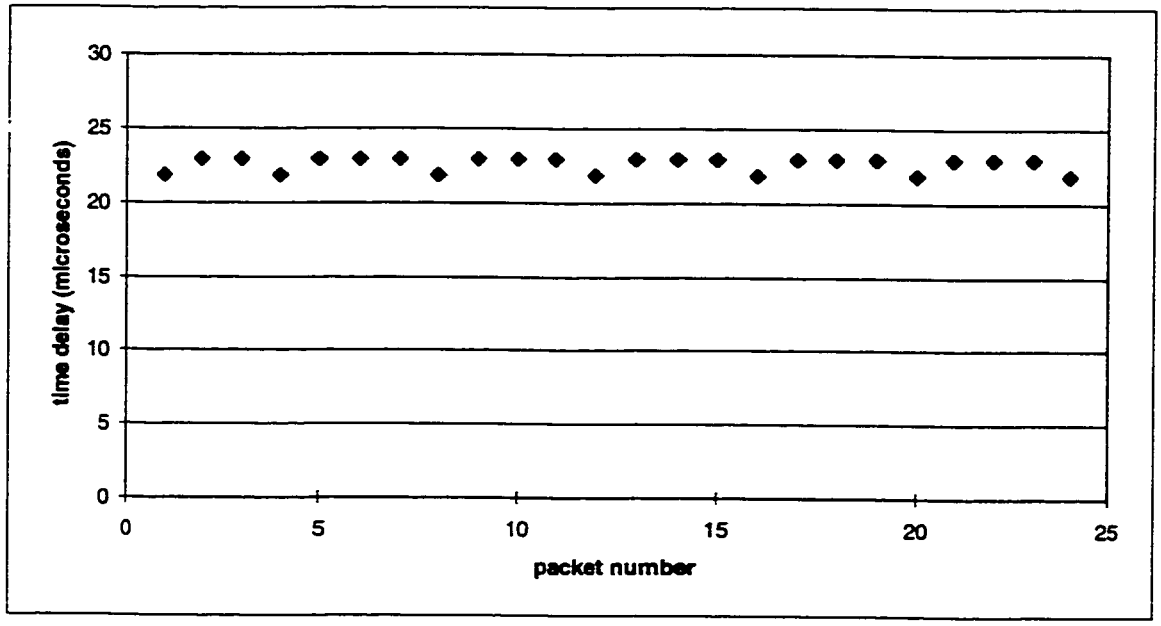


Figure 6-8 : time delay from station 0 to station 2 in scenario 3

Figure 6-8 shows that there are only a number of delays still have the value of 21868 ns. Most packets going to station 2 experienced 22868 ns delay. This is the direct result of the fact that one more station, station 2, is transmitting on the ring. As more stations on

the ring are transmitting, the abundance of empty packets become less. Therefore, from a individual station perspective, it has to wait a little longer to see a empty packet comes in on the ring. In the case of scenario 3, station 2 waited one extra bus clock cycle to receive an empty packet. This caused most of the delay recorded to be 1000 ns longer than those in scenario 1. This 1000 ns represents roughly a 5% increase in end to end delay.

In Figure 6-7, we can see that some delay values are the same as or very close to those in scenario 1 while others have relatively big increase. The smallest delay is consistent with the results from scenario 1. They represent the time delay where the video data packets had no collisions. In scenario 3, both station 2 and station 0 sent packets to station 1 at the same time. Sometimes, the data packet from station 2 arrives at station 1 before the video data packet from station 0 does. In this case, the receiving station has to finish the reception of the entire Ethernet that arrives first before receive any other packet. This put the video data packet on hold. The period of waiting depends on the number of bytes left for receiving in the packet that arrives first. The result shows that, in scenario 3, the longest wait is 30600 ns, which is represented by the largest delay of 51468 ns. This gives a 147% increase in end to end delay comparing to those recorded in scenario 1. Since the Ethernet data packet size is quite short, due to the very short FIFO used in this implementation, the waiting period would be longer when long Ethernet packet was received first, hence larger percentage increase in overall end to end delay.

From the network simulation results, presented in section 5.3, we noticed that when there was collision, the end to end delay increased up to 24 μs. It is a 11900% increase from the 0.2 μs minimum. A 10 μs delay gives a 4900% increase in end to end delay. Furthermore, since the delay after collision is random, the maximum may very well exceed 24 μs. The above observation indicates that the LAN on a chip architecture did improve, to some degree, the performance in terms of reducing and controlling the jitter. Even if we take into account the longer Ethernet data packet, the LAN on a chip architecture still offers us hope that we can get a delay well below those resulted from the Ethernet bus architecture. Theoretically, the longest wait would happen if the longest Ethernet data packet arrived

first at the receiving station. This puts a upper limit, not a random number, on jitter for the LAN on a chip architecture.

6.4 Summaries

The results, from the simulation of the FPGA implementation of the LAN on a chip architecture with three scenarios, show that there was no collision when each transmission has different destination station. When more than one transmission has the same destination station, collision occurred. The time delay caused by the collision varies, but is not random, with the maximum delay, in theory, equals to the time passing the longest Ethernet data packet on the ring. The software simulation shows that, comparing to the Ethernet common data bus architecture, the LAN on a chip architecture causes less time delay increase. This makes the LAN on a chip architecture a possible candidate for the future LAN architecture for real-time applications.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, a novel LAN on a chip architecture is implemented. The aim of this implementation is to investigate the performance of this new architecture on reducing and controlling the jitter experienced by the real-time traffic on local area network. The design was written in VHDL and was synthesized with the Altera' s EPF10K100GC503 FPGA as its target device. The FPGA device was simulated with VHDL simulator using software model from Synopsys.

The design borrowed the P-Bus architecture proposed by Dr. Wayne Lucks. The hardware of this new architecture is very simple and also efficient in passing Ethernet packets. Due to its simplicity, the hardware can operate at a relatively high speed.

In the simulation, the time delay was measured, between the input of the input FIFO of the source station and the output of the output FIFO of the destination station, for the Ethernet packets passing through the LAN. The results show the LAN on a chip

architecture improved the transmission performance in terms of reducing jitter in real-time applications. More than one transmission can take place at the same time on the ring without collision, when these transmissions have different destination stations. This reduces the time delay and minimizes the jitter. Collision only happens when more than one transmission has the same destination station. In the case of collision, the maximum delay one can get is the time needed to transmit the longest Ethernet packet on the LAN.

7.2 Future Work

The LAN on a chip architecture, proposed and implemented in this thesis, is a proof of concept that this architecture can significantly reduce and control the jitter for real-time LAN traffic. The preliminary simulation results demonstrated that this can be achieved efficiently in real-time with simple, dedicated hardware.

There are several factors that limit the maximum operating speed of the ring bus. Among them, the technology used for the Altera FPGA device and the external FIFO puts a minimum limit on the propagation delay required for each component. As a result, the maximum clock frequency at which the ring bus can operate could not be further increased. In future research, the LAN on a chip design can be implemented on an ASIC (Application Specific Integrated Circuit) device to greatly improve the speed of the circuit. Also, some modules or components can be further optimized to gain further efficiency and, hence, improve the speed.

The FIFO used in the test implementation is short, which limited the Ethernet packet used in the testbench to short packets as well. This made it difficult to test performance of the LAN on a chip architecture in situations where long packet is transmitted or bursty data traffic occurs for short period of time. In future research, larger size FIFO can be used to simulate situations more closer to real life.

Hardware prototype can also be produced in future research so that the performance can be tested in real hardware implementation.

References

[1] William Stallings, " Data And Computer Communications" , Prentice Hall, 1994

[2] Wayne M. Loucks, " Fermtor: A Flexible Extendible Range Multiprocessor" , Ph.D. Thesis, University of Toronto, 1980

[3] J. Rumbaugh, " A Data Flow Multiprocessor" , IEEE Trans. Computer, Vol. C-26, No. 2, Feb. 1977, p138-146

[4] J.B. Dennis, " The Varieties Of Data Flow Computers" , Proceedings of the 1st International Conference on Distributed Computing Systems, Oct. 1979, p430-439

[5] I. Watson and J. Gurd, " A Practical Data-Flow Computer" Computer, Vol. 15, No. 2, Feb. 1982, p51-57

[6] L. Stringa, " EMMA: An Industrial Experience On Large Multiprocessing Architectures" , Proceedings of the 10th Symposium on Computer Architecture, June 1983, p326-333

[7] E.P. Farrel, N. Ghani, and P.C. Treleaven, " A Concurrent Computer Architecture and a Ring Based Implementation" , Proceedings of the 6th Annual Symposium on Computer Architecture, April 1979, p1-10

[8] Wayne M. Loucks, Zvonko G. Vranesic, " FERMTOR: A Flexible Extendible Range Multiprocessor" , Proceedings of the Canadian Information Process Society, Victoria, B.C., Canada, 1980, p134-149

[9] W. M. Loucks, W. J. Jager, " P-Machine: A Hardware Message Accelerator For A Multiprocessor System" , Proceedings of the International Conference on Parallel Processing, Piscataway, NJ, USA, 1987, p600-609

[10] J.S. Rose, " An Implementation of FERMTOR: A Flexible Extendible Range Multiprocessor" , M.A.Sc thesis, University of Toronto, 1982

[11] High-Performance FIFO Memories Data Book, Texas Instruments, Oct. 1997