ECED 4902
Senior Year Project
Design and Implementation of a Cartesian Router

Students:
Edward A. Wilkinson
B00082703

Karen MacGillivray
B00134351

External Supervisor:
Dr. Larry Hughes

Internal Supervisor:
Dr. Jacek Ilow

Company:
Department of Electrical and Computer Engineering
Dalhousie University

Date Submitted:
2001/12/03

Abstract

The goal of this project was to design and implement a Cartesian router using hardware-based decision logic. This report serves as a summary of the work completed on the project over the period September 9, 2001 to December 2, 2001.

During this time, a hardware logic design was produced. Several implementation methods were considered, including discrete logic gates, FPGAs, and ASICs. The discrete logic gates were originally chosen to minimize the learning curve and maximize modifiability. Unfortunately, the discrete logic router proved to be too large for two students with full course-loads to complete in the time frame allotted. Thus, FPGAs were chosen to implement the design. However, the switch was not made in time to produce an FPGA router. Router construction efforts will continue in the time between exams and the presentation on December 19, 2001, and the components that have been successfully produced will be demonstrated during the presentation.

The contents of the report are as follows. It begins with a brief overview of conventional routing and Cartesian routing. It then presents a summary of the logical design of the router, and the options considered for physically implementing the router. This is followed by a description of the problems involved in the implementation process. Finally, the report presents recommendations for projected changes to the router, should the project be resumed in the future.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

Cartesian routing is a new and unique routing architecture that is quite different from existing routing methods. The key distinction of Cartesian routing, when compared to conventional routing, is its independence from routing tables[2].

In general, routers are used to connect networks. A conventional router uses routing tables to keep track of which networks are connected to its ports. The conventional router receives a packet, gets the packet's destination address, and then searches its routing table to determine which port is associated with that destination address[6].

The conventional router can only know where a packet is going by consulting its routing table. Thus, the conventional router uses a routing table to determine whether to keep, forward, or discard the packet and, if forwarding the packet, to which router the packet should be forwarded[6].

Cartesian routing avoids the use of a routing table by imposing a strict addressing scheme upon routers and networks that is dependant upon their physical location. By comparing the packet's destination address with its own location address, the router knows whether to keep, forward, or discard the packet, and which way to forward it, all without the use of a router table. What this does is reduce the router's decision to a simple comparison of the packet destination address and the router address, eliminating the need to maintain and search a database of network state information[2].

The goal of this project was to design and build a Cartesian router, with some restrictions. A preliminary Cartesian router, the Estabrooks-Poirier Router, was developed last year. It employed a software-based decision engine[1]. Although simple and modular in design, this microprocessor-based solution proved to have a slower performance than desired, and led our external supervisor, Dr. Larry Hughes, to the conclusion that the router would be faster if implemented purely in hardware.

Thus, we were charged with the task of developing a Cartesian router with a hardware-based decision engine. The Cartesian router was required to be as fast as possible, while still remaining economically feasible. These restrictions were the only constraints placed on the final product.

2.0 Background

The background section contains detailed descriptions of conventional routing, Cartesian routing, and the Estabrooks-Poirier router.

In section 2.1, we define conventional routing and the nature of conventional routers. Routing tables are explained, and their performance characteristics are briefly analyzed.

In section 2.2, the concept of Cartesian routing is introduced and summarized. The reader is directed toward Dr. Hughes' website, a more comprehensive source of information on Cartesian routing.

Finally, in section 2.3, we present a summary of the Estabrooks-Poirier Router. The Estabrooks-Poirier router is an existing implementation of a Cartesian router. It was designed and built as a senior-year project. It is explained below how we were able to use their experience to enhance the performance of our Cartesian router. We also explain the key difference between our router and the Estabrooks-Poirier router.

2.1 Conventional Routing[6]

As mentioned in the introduction, routers are used to connect networks, a process called internetworking. A conventional router talks to its associated networks through ports. The router uses routing tables to track which networks are connected to its ports. It receives a packet, gets the packet's destination address, then searches its routing table to determine which of its ports is associated with that destination address.

Routing tables therefore store the address and associated port number of every network for which they route traffic. In addition to the address and associated port number, the routing table also must store the range of machine addresses that are part of the network, since the destination address is a machine level address. This is accomplished using a netmask, a process that is explained in greater detail in below.

Before the packet can be forwarded to the appropriate port, its destination address must be compared to the values in the routing table. There is an average search time associated with the routing table. In general, the average search time increases as the table length increases.

The general structure of a routing table is shown below in Table 1.

When a packet arrives at the router, the router does two things: it gets the packet's destination address, then compares it to the addresses in the router table until it finds a match, at which time it forwards the packet to the appropriate port. Since the destination address is usually the address of a certain machine on a network, it must be masked to determine which network it belongs to. This is the purpose of the netmask listed in the sample routing table.

Table 1: The Structure of a Routing Table[1]

Network Address	Network Netmask	Port
100.200.123.0	255.255.255.0	0
24.100.100.0	255.255.255.0	1
12.230.20.0	255.255.255.0	2
101.100.233.0	255.255.255.0	0

So for each entry in the router table that the destination address is compared to, two things must occur. First the destination address must be masked with the netmask, and second, the resulting value must be compared to the network address. If they are equal, the destination machine is on that network. The destination address is compared to each network address in the router table until a match is found, or it hits the end of the table. For a long enough routing table, this process may be time-consuming and affect the performance of the router.

2.2 Cartesian Routing[2][3]

The essence of Cartesian Routing is the imposition of a strict addressing scheme on all routers. The addresses of the routers in a network are based on the router's physical locations, with the latitude given first and the longitude given second. Since all addresses are based on physical locations, all a router has to do when a packet arrives is compare the packet's destination address with its own and then forward it in the right direction (be it up, down, left, or right). Thus, the need for router tables is eliminated, reducing the number of address comparisons to one.

Of course, there are potential complications for this addressing scheme. If a packet is sent to an address where no router exists, the two routers on either side of the address could theoretically bounce a packet back and forth forever. This complication is handled by adding the stipulation that if a packet is to be forwarded out the same port that it arrived from, then that packet should be discarded.

A Cartesian Network is implemented with two types of routers: a Collector Router and an Arterial Router. The main difference between these two routers is that the Collector Router has two external ports (generally east and west) and an internal port, while an Arterial Router has more than two external ports. The average Arterial Router has four external ports (usually north, south, east, and west) but it can have as few as 3 or as many as 6 in a 2D scheme.

The other difference between a Collector and an Arterial Router is how they handle latitude. When a Collector Router receives a packet whose destination address contains a different latitude it assumes the packet is moving in the correct direction and simply

[1] For illustrative purposes, the network addresses and network netmasks used are formatted according to the IP network protocol.

forwards it. The Arterial Router, on the other hand, will make no such assumption and will attempt to forward the packet in the correct direction. If the latitude of the destination address matches the latitude of the router, then both routers will do a longitude comparison and attempt to forward the packet in the proper direction (be it east, west, keep, or discard).

To find out more about Cartesian routing, visit Dr. Hughes' website at http://www.dal.ca/~lhughes2/cartnet/index.html.

2.3 The Estabrooks-Poirier Router[1]

In the fall term of 2000 Kevin Estabrooks and Pascal Poirier designed a prototype Cartesian router. They successfully managed to build a collector router that was capable of routing data at 9600 bits per second. The Estabrooks-Poirier router was designed so that each port consisted of an Atmel AT90LS8535 RISC microprocessor, a 1K X 9 Asynchronous FIFO, and tri-state buffers.

The router worked as follows. First, data would arrive at the UART of a processor. The processor would decide which way to send the packet using software. The packet would then be sent to a tri-state buffer until such time as it could be loaded into the FIFO for the destination port (which can only accept one packet at a time). Packets in the FIFO would then be transmitted to the destination Atmel microprocessor when it is no longer busy. The destination processor would then transmit the packets to the next router.

The Estabrooks-Poirier router was built and was proven to successfully route packets. However, the UART on their chosen microprocessor limited the speed of their router to 9600bps. For this reason, Dr. Hughes requested that we implement the router logic in hardware, in hopes of creating a faster router.

3.0 Design and Implementation

"After discovering that FPGAs and ASICs are too expensive, we tried to see if we could make a serial hardware based router with only a microprocessor to help direct traffic. This did not go very far in the design process because of the level of complexity. Using individual components would have complicated the design and make the router unreliable at high speeds because of the wiring required to connect the components. The same wiring problem applied FPGAs. FPGAs would have worked but much of the logical processing would have to been done in smaller chips or PLDs. The design would have become too large and complicated. Timing and cross talk issues would have become much too prevalent to pick this design".

-Kevin Estabrooks and Pascal Poirier, Simplified Cartesian Routing, 2000

When Dr. Hughes requested that we implement the router in hardware, we decided to test the Estabrooks-Poirier theory (quoted above). The purpose of redesigning it with hardware was to increase the speed of the router by eliminating the overhead operations of a microprocessor. Not having much experience in digital hardware design, our preliminary goal was to develop a hardware design for a Cartesian router that could be built using any number of hardware implementations.

In other words, our goal was to make the actual design as independent from the hardware as possible, so the number of design modifications required for any particular implementation method would be minimal. After completing the design we would then have to choose a method with which to create a proof-of-concept prototype. The prototype would subsequently be created, after which we would make a recommendation on how to implement the final product.

3.1 Theoretical Design

The original intent of the project was to create an Arterial Cartesian Router. In our early design discussions, it became apparent that a Collector Cartesian Router could also be implemented using the component modules that were designed for the Arterial Router. This section of the report begins with an overview of the internal workings of these two types of Cartesian routers.

For both the collector and arterial routers, section 3.1.1 illustrates the flow of information between the ports.

Section 3.1.2 takes a closer look at a generic port. The generic port is broken down into its component modules --- the Packet Detection Module (PDM), the Decision Making Module (DMM), the Incoming Packet Storage (IPS), and the Outgoing Packet Storage (OPS). Each port on the router handles the packets uniquely, because of the port-specific nature of Cartesian routing.

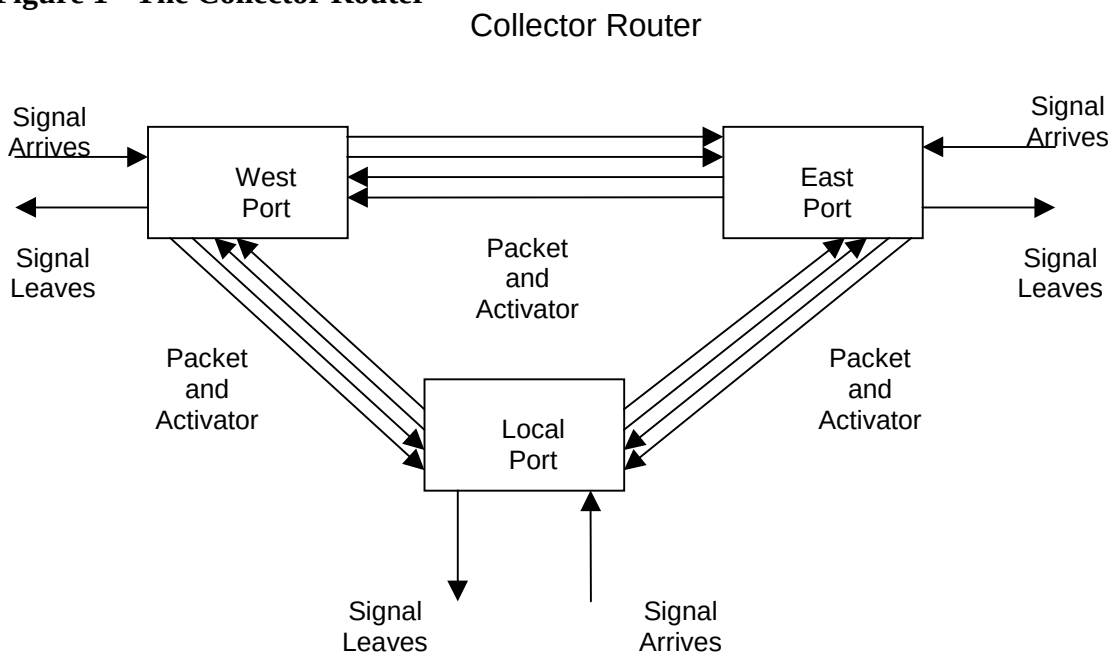
In sections 3.1.3-3.1.7, the components of the generic port are explained in great detail. The hardware level design figures for each module are provided, along with supportive information where appropriate (state diagrams and logic tables).

3.1.1 Overall for the Router

The first step in the design process was to divide a router into smaller components. We chose to divide the router into its component ports, which would then send packets back and forth to each other as needed. The router must be able to handle packets coming in from multiple ports simultaneously. No central control is provided in our design because processing multiple packets is faster if each individual port contains its own control logic. Thus, the individual ports should be able to make packet routing decisions on their own.

We started by dividing the collector router into its component ports, as it is the simplest Cartesian router. The Collector router design is shown below in Figure 1.

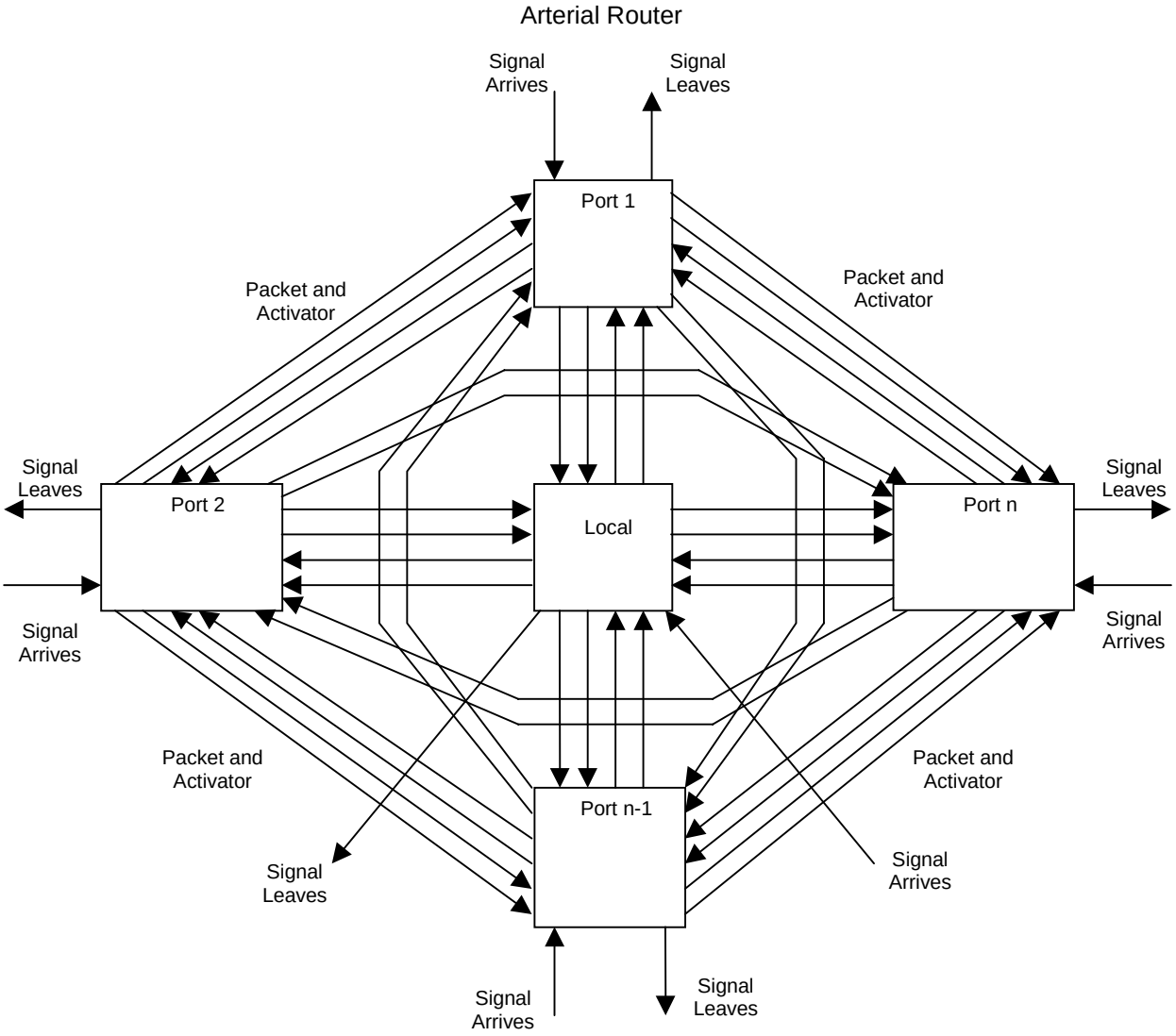
Figure 1 - The Collector Router



Signals arrive and leave at each port. The ports send packets to the other ports as appropriate, along with an activator signal that tells the other port to start reading. This activator signal allows us to forgo a centralized control system, thus reducing overhead.

Next we divided the Arterial router into its component ports. The arterial is more complicated, as it can have a differing number of ports. This has been represented in the diagram by using ports n-1 and n. As before, signals arrive and leave the router from the various ports. The various ports again send packets and activators to the other ports as appropriate.

Figure 2 - The NEWS Arterial Router

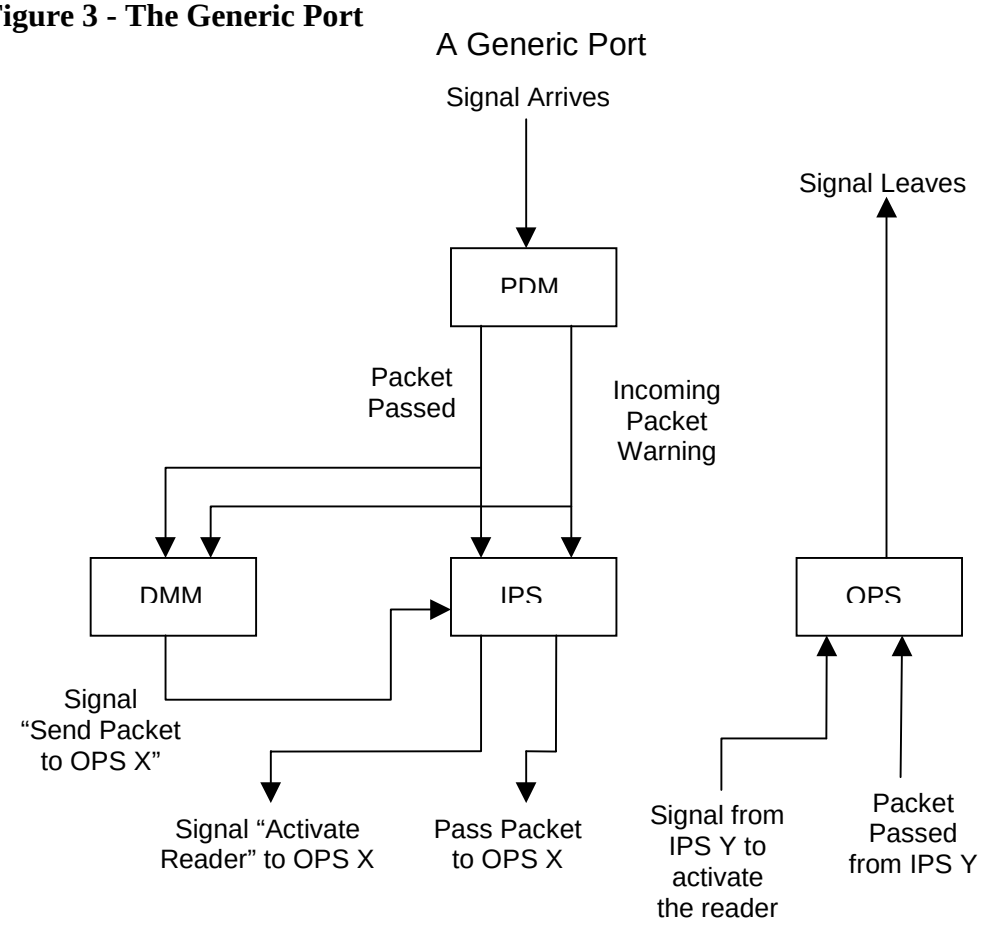


3.1.2 High Level Port Design

The high level port design contains four modules:

1. The Packet Detection Module (PDM), which detects the incoming packet and activates other modules so they can deal with the packet.
2. The Decision Making Module (DMM), which decides what should be done with the packet.
3. The Incoming Packet Storage (IPS), which stores the incoming packet until the DMM decides what to do with it.
4. The Outgoing Packet Storage (OPS), which stores packets from other ports that are to be sent out from this port.

Figure 3 - The Generic Port



The information flow for the port is as follows:

1. A signal arrives and is detected by the PDM.
2. The PDM sends an "Incoming Packet Warning" message to both the DMM and the IPS. This warning causes the DMM to start making the routing direction decision and the IPS to start recording the packet so none of it is lost.
3. The DMM eventually decides where the packet should be sent and then sends the destination port to the IPS.
4. The IPS sends an activator signal and the packet to the OPS of the proper port, discarding the packet if need be.

The local OPS works adjacent to but independently of the other modules in the port. When it receives an activation signal from an IPS it begins recording the packet. The packets received from the various ports are then transmitted in a sequential fashion to avoid starvation.

3.1.3 The PDM (Packet Detection Module)

The Packet Detection Module (PDM) detects an incoming packet, and enables the DMA to process the packet. It uses a string of six logical ones as both the Start Of Packet (SOP) marker and the End of Packet (EOP) marker. The IN line idles on zero. When the PDM

detects a SOP, it sets the latch to indicate that the port is receiving. It then continues to receive until an EOP arrives, at which time the latch is reset, and the port stops receiving.

The state of the PDM is determined by the output values X and Q (the latch), as shown in the state diagram below.

Figure 4 - The Packet Detection Module

The Packet Detection Module

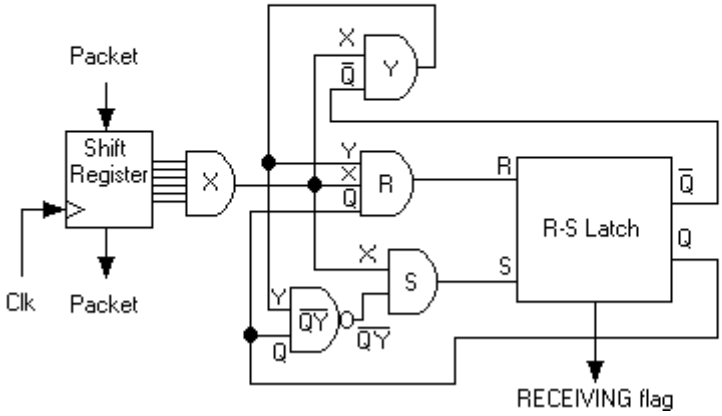
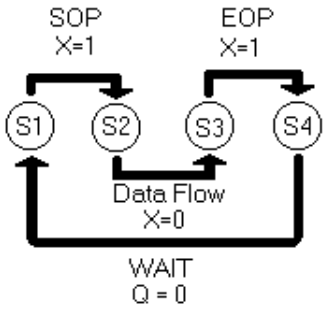


Figure 5 - The Packet Detection Module State Diagram

The Packet Detection Module
State Diagram



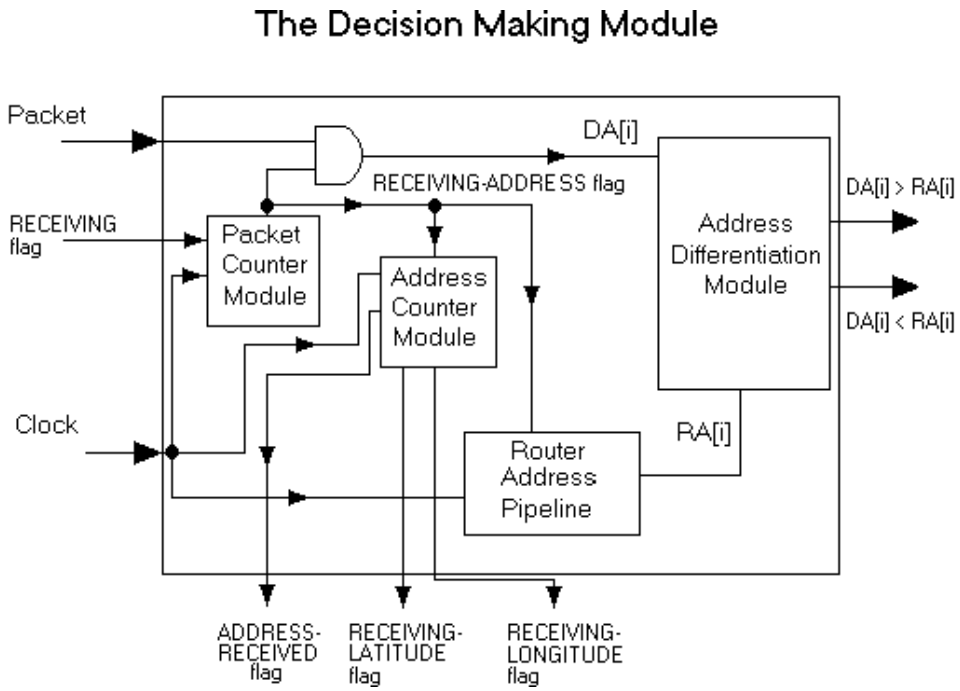
S1	Q = 0 X = 0	WAIT
S2	Q = 0 X = 1	SOP
S3	Q = 1 X = 0	EOP
S4	Q = 1 X = 1	DATA

3.1.4 The DMM (Decision Making Module)

The DMM strips the destination address from the packet then compares it to the router's address. Based on the result of the comparison, it tells the IPS to either keep, discard, or forward the packet to the appropriate port.

The DMM has four component modules. They are the Packet Counter Module (PCM), the Address Counter Module (ACM), the Router Address Pipeline (RAP), and the Address Differentiation Module (ADM).

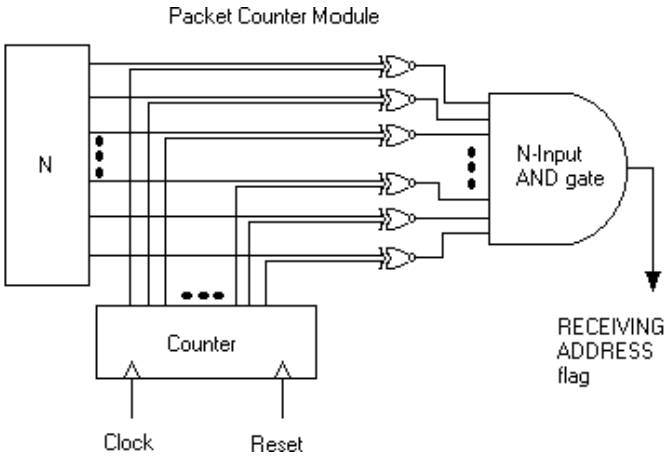
Figure 6 - The Decision Making Module



3.1.4.1 The Packet Counter Module (PCM)

The PCM strips the destination address from the packet. It counts the incoming packet bits and sets the RECEIVING-ADDRESS flag when the first bit of the address is read.

Figure 7 - The Packet Counter Module

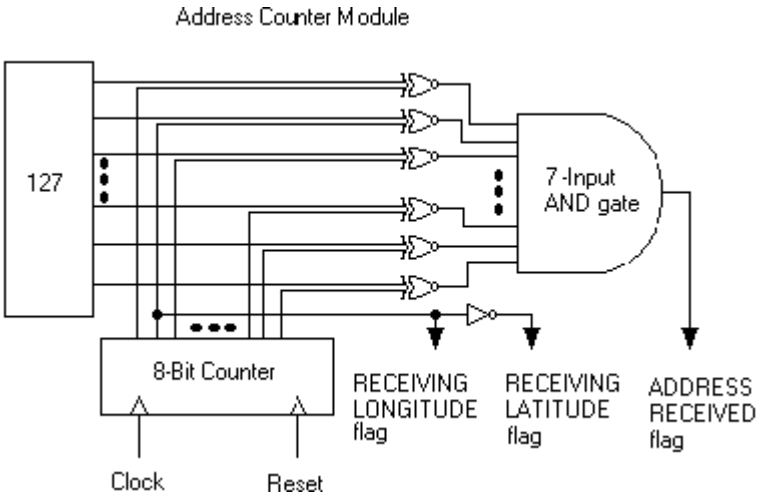


3.1.4.2 The Address Counter Module (ACM)

The ACM keeps track of the number of address bits that have been read. It indicates which portion of the address is being received (latitude or longitude), and, if the entire address is received, it sets the ADDRESS-RECEIVED flag. When the ADDRESS-RECEIVED flag is set, the IPS is told to keep the packet.

The RECEIVING-LATITUDE flag is set when the first 64 bits of the address are being received. The RECEIVING-LONGITUDE flag is set and the RECEIVING-LATITUDE flag is reset when the 64^{th} to 128^{th} bits are being received. The port uses these RECEIVING flags and the results from the DMM to determine which line to set HI on the IPS.

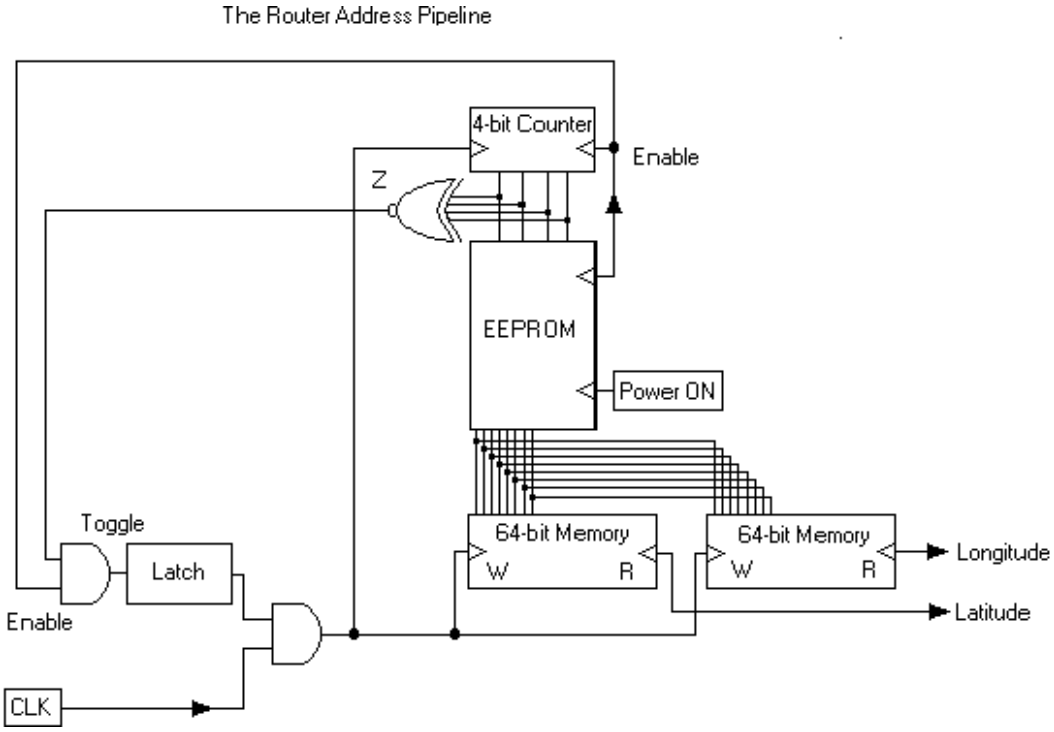
Figure 8 - The Address Counter Module



3.1.4.3 The Router Address Pipeline (RAP)

The RAP stores the router address and pipes it out serially so that it can be compared with the incoming destination address. The router address is a static value, and is preferably kept in non-volatile memory for long-term storage (for example, ROM, PROM, EEPROM). Because programmable ROMs are usually quite slow (access times of 350ns-450ns are common), the router address will likely be stored in a faster short-term memory for runtime access. For a discrete component router, the short-term memory could be RAM or a shift register. For an FPGA implementation, the router address may simply be stored in a register. For the sake of keeping the logical design generic, the logical circuit below indicates the short-term memory as two 64-bit memory modules.

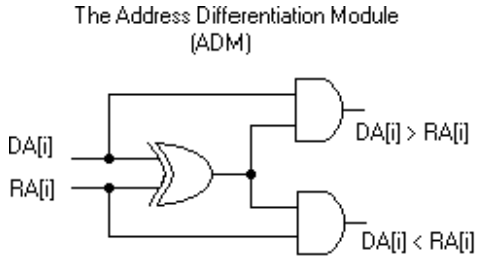
Figure 9 - The Router Address Pipeline



3.1.4.4 The Address Differentiation Module (ADM)

The ADM simply compares the i^{th} bit of the destination address (DA[i]) to the i^{th} bit of the router address (RA[i]), as the destination address is read into the DMM. If DA[i]doesn't equal RA[i], the ADM tells the Incoming Packet Storage (IPS) module what the result of the comparison was by setting the appropriate line (DA[i] less than RA[i], or DA[i] greater than RA[i]) to logical one. Precise timing is essential.

Figure 10 - The Address Differentiation Module



ADM Truth Table

DA[i]	RA[i]	DA[i] < RA[i]	DA[i] > RA[i]
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

3.1.5 The IPS and Sub-Components

The Incoming Packet Storage module is used to store incoming packets until the DMM decides on a direction to send the packet. Once the decision is made and sent to the IPS, the IPS transmits an activator signal and the packet to the OPS on the chosen port. The various components and signals of the IPS can be seen in Figure 11.

The logic diagram for the IPS is included on the next page. It was drawn for n ports, so it could serve as the design as the IPS for either a Collector or Arterial Router. In the case of the collector router it will have the port 1, keep, and discard lines. In the case of the arterial router it will have the keep line, the discard line, and as many port lines as needed for the particular router.

3.1.5.1 The FIFO

The core of the IPS is the FIFO that is used to store the incoming packet. Early on we learned that many commercial FIFOs come with parallel inputs rather than serial inputs. To handle the eventuality that such a FIFO might be used, a FIFO data converter was added to both the input and output of the FIFO to convert the data from serial to parallel.

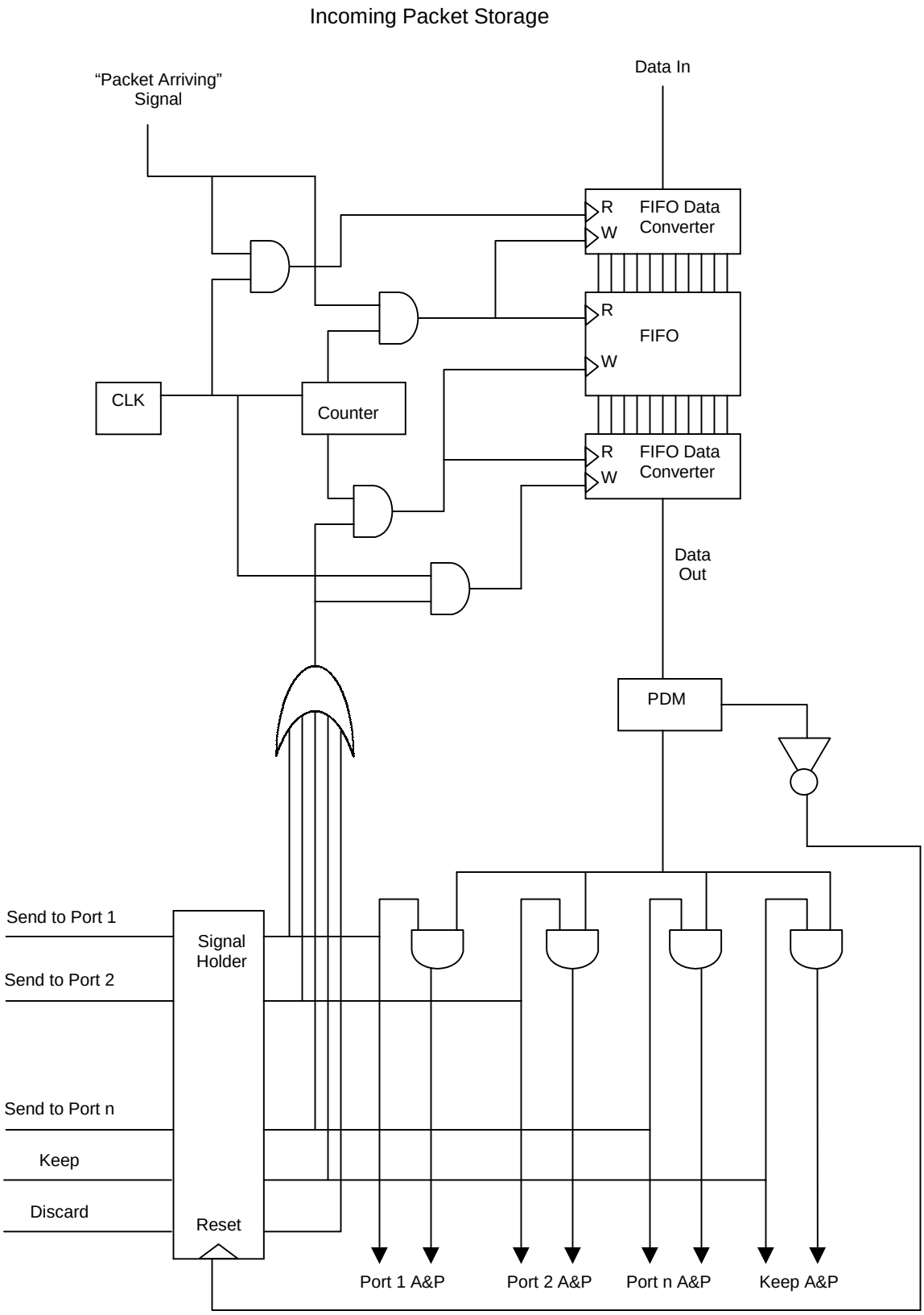
The serial-to-parallel conversion requires that different clock signals be used to handle the serial and parallel readings and writings, as several serial read-and-write operations must occur for every one parallel read-and-write operation. This differential clocking is handled by sending a clock signal into a counter. The counter is of a size that matches the number of parallel entries, and outputs a signal when that many clock cycles have passed. Thus, the counter serves as a stepped-down clock for the parallel read-and-write operations. The regular clock signals activate the reading function of the input FIFO data converter and the writing function of the output FIFO data converter. The counter clock signals activate the writing function of the input FIFO data converter, the reading and writing functions of the FIFO, and the reading function of the output FIFO data converter.

It should be noted that if a serial FIFO can be found then the design can be simplified by removing the FIFO data converters, the clock's counter, and the gates associated with both. A similar simplification could be made by simply using a single pin of a parallel FIFO, essentially converting it into a FIFO with less memory.

3.1.5.2 Receiving and Transmitting (IPS)

To actually receive messages, a "packet arriving" signal must be received by the IPS. The "packet arriving" signal line is fed into AND gates that combine it with the clock and counter signals that go to the input FIFO data converter and the read function of the FIFO. When there is no "packet arriving" signal the output of the AND gate is 0 and the FIFO does not read anything. When the packet arrives the clock and counter signals are

Figure 11: The Incoming Packet Storage Module



output by the AND gate, causing the FIFO to read incoming data at the rate determined by the clock. The data is then held until it is time to transmit.

Transmission of the packet to the OPS on the appropriate port begins when a signal is received from the DMM. There is a line from the DMM for every decision that might be made; one for each of the other ports (including the internal 'keep' port), and one for the discard. To inform the IPS of its decision, the DMM simply lights up the appropriate line.

The signal from this line is sent to two places to trigger the writing process. First, the signals from all of the DMM lines are sent to a combinatory OR gate. Since only one of the lines should be high, this should have the effect of piping all of the line outputs onto the same path. The OR output is then piped into AND gates along with clock and counter signals. These AND gates are connected to the write function of the FIFO and the output FIFO data converter. When the output from the OR gate is high (i.e. when the DMM has made a decision) the AND gates allow the clock and counter signals to pass to the FIFO, which in turn causes data to be transmitted from the FIFO.

The second place the signal is sent to is a series of AND gates, with each line being sent to its own AND gate. Thus, there is one AND gate for each OPS that the packet could be sent to. The signal from each line is ANDed with the output from the FIFO, so the packet will pass through the AND gate that has received the active signal. This effectively routes the signal to the OPS of the appropriate port.

In addition to allowing the packet to pass through the AND gate, this signal is also transmitted to the OPS of the appropriate port alongside the packet. Thus, this signal also serves as the activator for the OPS.

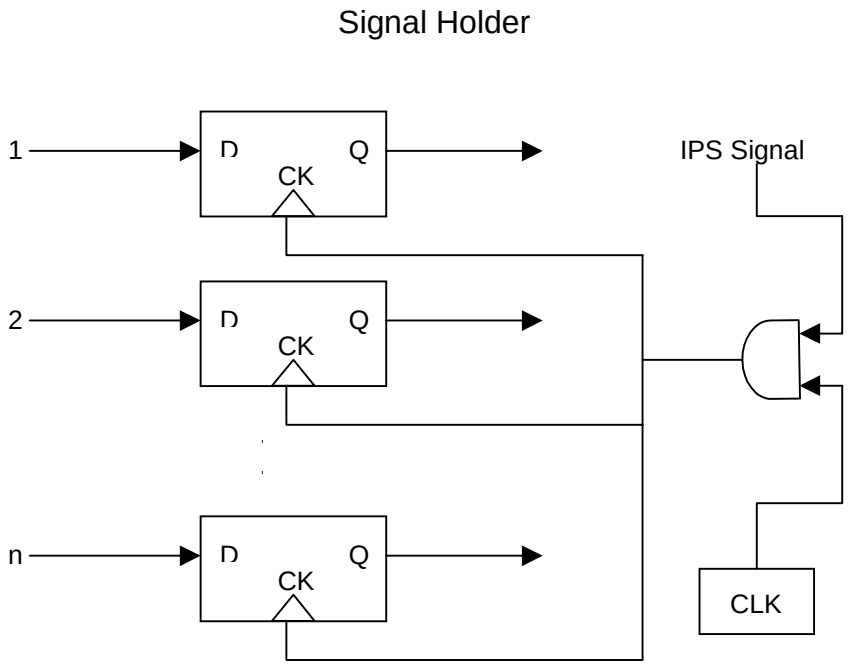
3.1.5.3 The Signal Holder

In order for the above logic to work the signal from the DMM must be held high until the packet has finished transmitting. Thus, a signal holder was developed to receive the decision signals from the DMM and transmit them to the rest of the IPS until the packet has finished transmitting. The signal holder consists of one D flip-flop for every decision line coming from the DMM (the exact number of which will depend on the number of ports in the router). When a reset signal is received from the IPS it is fed into an AND gate along with a clock signal. The reset signal allows the clock signal to pass through the AND gate and trigger the D flip-flops, which will then record what's on the lines. If one of the lines is high the IPS will stop sending the reset signal. A diagram of the signal holder is included in Figure 12.

Of course, for this signal holder to work a means had to be found to detect the end of packet. This is done by using a PDM between the FIFO output and the routing AND gates. If the DMM is not currently sending a decision then all the lines of the signal holder will go down and the FIFO will stop writing. This will cause the shift register to retain its 6 ones, thus causing the reset signal to be sent over and over again. This

process will continue until the DMM sends another decision, which will cause the FIFO to start writing and thus the shift register to stop having 6 ones. This in turn will keep the signal holder from being reset until the end of packet is again reached.

Figure 12 - The Signal Holder



3.1.6 The OPS and Sub-Components

The Outgoing Packet Storage Module is used to store outgoing packets and then send them one at a time. The various components and signals of the OPS can be seen in Figure 13.

3.1.6.1 The OPS FIFOs

The OPS is designed to have one FIFO for the IPS module of every other port on the router. The purpose of using FIFOs for every IPS module is to handle competition for a particular port. With FIFOs available for every IPS module more than one IPS can send a packet to an OPS module at the same time without having to wait. Waiting schemes were also considered to reduce the number of FIFOs needed, but the acknowledgments (and associated storage and logic) required to facilitate such a scheme would have greatly increased the complexity of the circuit.

It should be noted that the OPS design drawing has been made with n FIFOs, so it can be used as both a collector router design and an arterial router design. The collector router

would have two FIFOs per OPS, while the average arterial router (with north, south, east, and west ports) would have four FIFOs per OPS.

The OPS FIFO is identical to the IPS FIFO described in section 3.1.5.1, with the exception that we will need to know whether or not the FIFO is empty. As such, any FIFO that we use here must have an "Empty" indicator that either sends a high signal or makes some other indication that can then be converted into a high signal.

3.1.6.2 Receiving and Transmitting (OPS)

When a packet arrives from an IPS, it is accompanied by an activation signal. An AND gate is used to combine the activation signal with the clock and counter signals that go to the input FIFO data converter and the read function of the FIFO. The activation signal causes the AND gate to pass the clock and counter signals to the FIFO, allowing the packet to be recorded at the rate determined by the clock. The data is then held until it is time to transmit.

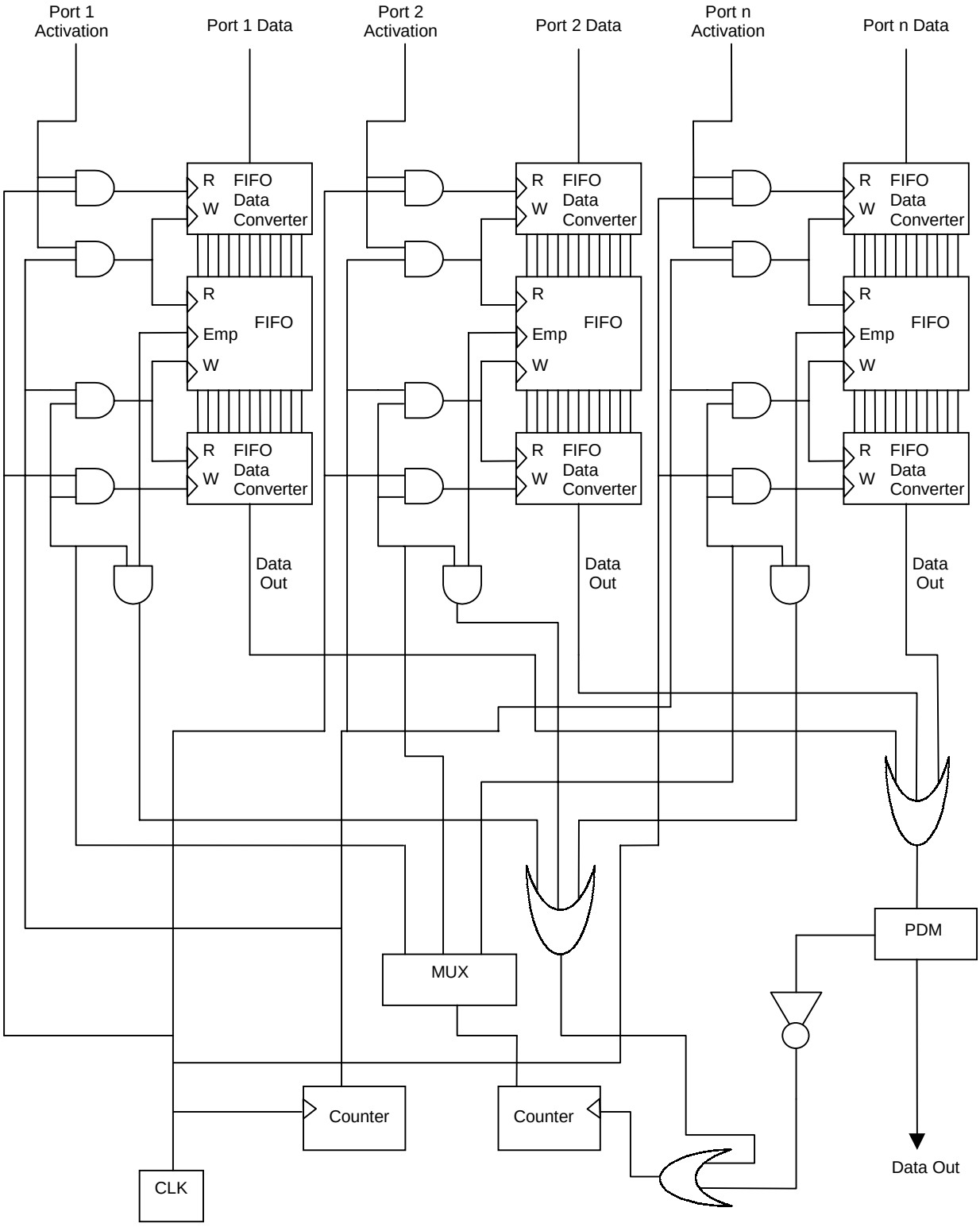
Each FIFO takes turns transmitting. The MUX has a line for every FIFO. Each of these lines is fed into the AND gates that control the write function of the FIFO, the read and write functions of the output FIFO data converter, and the "Empty" indicator of the FIFO. The MUX only lights up one line at a time, so that only one FIFO can write at any one time. As only one FIFO can write at any one time, all of their outputs are fed into an OR gate to get them all onto the same output line.

The MUX switches lines based on the output of a counter that is connected to it. Each output line on the MUX corresponds to a number generated by the counter. For example, if the counter outputs a 0 the 1st MUX output line is activated, and if the counter outputs a 1 the 2nd MUX output line is activated. The counter is set up to loop back to 0 after it has iterated through all of the output lines of the MUX. For example, if the MUX has four output lines (for the four FIFOs required for an average arterial router), then the counter will count from 0 to 3.

The counter that runs the MUX is to be incremented under two circumstances: one, if the end of a packet is reached, and two, if the FIFO that is currently being checked is empty. Signals indicating the truth of these conditions are fed into an OR gate that is connected to the counter. When either condition is true the output of the OR gate goes high and the counter is incremented.

The OPS uses a PDM to determine if the end of packet has been reached. The output from the PDM is sent through an inverter so it will be low when a packet is passing through and high when the packet has ended. This signal is then fed to the OR gate that controls the MUX counter.

Figure 13: The Outgoing Packet Storage Module



To determine if a FIFO is empty, the "Empty" signal is sent from the FIFO to an AND gate. The MUX signal is also sent to the AND gate, so the empty signal is only passed if the FIFO is the one that is currently writing. Thus, only one FIFO can send out an "Empty" signal at a time. The "Empty" signal lines from these AND gates are brought together onto the same line using an OR gate (which works, because only one of the "Empty" AND gates can be high at a time). The output from this OR gate is then fed into the OR gate that controls the MUX counter. If the "empty" output is high the MUX counter is incremented and the MUX activates the line for the next FIFO. This way the OPS always moves to the next FIFO when the current one is empty.

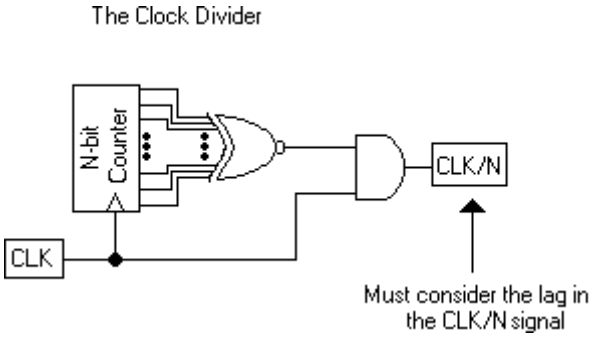
3.1.7 The Clock

There is a need for clocking in several modules. To maintain synchronicity, there is a central clock for the entire router. Since some modules require a slower cycle than other modules, in some cases the central clock signal must be divided down to the desired clock cycle. All other clocking signals are derived from the central clock. Logically, the clock division is accomplished using a binary counter, a NOR gate, and an AND gate.

Important design considerations:

i. The maximum clock speed is that of the central clock.
ii. There will be lag in the CLK/N signal.

Figure 14: The Clock Divider



3.2 Implementation Options

This section contains descriptions of the different methods that were considered for implementing the router.

3.2.1 Discrete Component Router

The discrete component (TTL and CMOS components) implementation of the prototype router was considered because TTL/CMOS parts are inexpensive and easily available, and the implementation of a TTL/CMOS logic circuit has a minimal learning curve.

Except for the large memory components, all necessary TTL/CMOS components were found to be available from the Electrical Engineering lab technicians and cost on average $0.85 cents.[2] The large memory components (router address pipeline (RAP), the storage FIFOs, EEPROM) were found to be available in a variety of products of varied cost and availability. A comparison of the considered products for the large memory components is shown below in Table 2. A preliminary cost of the overall discrete component port is provided below, in Table 3.

Table 2: Preliminary Analysis of Large Memory Components[10][1]

Large Memory Component	Product		Cost	Availability
EEPROM Long Term Storage for RAP (1)	**2716**	**16k (2kx8bit) UV Erasable ROM**	**Free[3]**	On Hand
FIFO Queue for Packet Storage (4)	**IDT7200L**	**256x9 CMOS Asynchronous FIFO**	Unknown	In Stock at Inland Empire Components
	IDT7201LA	**512x9 CMOS Asynchronous FIFO**	$6.81	In Stock at Carleton-Bates
	IDT7202LA	**1024x9 CMOS Asynchronous FIFO**	Unknown	In Stock at Inland Empire Components
128-bit Memory Fast Pipeline for RAP (1)	**MC14562B**	**128-bit Serial-Out Shift Register (1200ns)**	$4.19	In Stock at ON Semiconductor
	74F189	**64-bit RAM (15ns)**	$2.53	In stock at Fairchild Semiconductor
	74F219	**64-bit RAM (20ns)**	$0.54	In stock at Fairchild Semiconductor

Table 3: Preliminary Cost Analysis of Discrete Component Port[10][11]

Component		Number of Units	Cost per Unit
7400	**Quad 2 I/P NAND gate**	1	$0.85
7402	**Quad 2 I/P NOR gate**	3	$0.85
7404	**Quad Hex Inverter**	7	$0.85
7408	**Quad 2 I/P AND gate**	9	$0.85
7411	**Triple 3 I/P AND gate**	3	$0.85
7430	**8-I/P NAND gate**	16	$0.85
7474	**Dual D-Type Flip-flop**	2	$0.85
7486	**2-I/P XOR gate**	1	$0.85
74136	**Quad 2 I/P XNOR gate (O.C.)**	34	$0.85
74139	**Dual 1-of-4 Decoder/Demultiplexer**	1	$0.85
74163	**4-bit Binary Counter**	7	$0.85
74164	**8-bit Serial-in Parallel-out Shift Register**	11	$0.85
74269	**8-bit Binary Counter**	1	$0.85
CCO-014	**100MHz TTL Clock Oscillator**	1	$4.00
	Total per Port:		$85.60

This was only a preliminary investigation of the requirements of implementing the router in discrete TTL/CMOS components, and so the numbers are approximate.

For a collector router, which has two ports, the part count is ~200, at a cost of ~$230 with the large memory components. For the Arterial router, which has four ports, the part count is ~500, at a cost of ~$600.

[2] Estimate provided by the lab technicians in the Dalhousie Electical Engineering Lab.

[3] Free because the lab technicians had one on hand. It's actually very expensive (~$45), so would only be useful as a part of the prototype, and should be replaced in a final design. It is only used because it is on hand, and free.

3.2.2 ASIC Router

One way to implement the router would be with an Application Specific Integrated Circuit, or ASIC. In this case we would input the design into a simulator first to make sure it worked. Then we would have to contact a company that burns ASICs and have them create the chips for us. Depending on the company, they might require as little as a description written in a hardware description language, or as much as a fully detailed specification.

The big advantage of using an ASIC is that it would be the fastest electronic solution. In addition, it would probably take up the least space. There are a number of disadvantages, however.

From our perspective the biggest problem is the learning curve. As students in the computer option we do not get to take the IC design technical elective. As such, we would, in effect, have to pick up another course to execute this properly. This problem would be reduced significantly if we could find a supplier that could build the chip using a hardware description language simulation of the circuits. Otherwise, we might have to sort out difficulties with unwanted capacitances and inductances produced in such a small circuit, and our experience with such matters at the microchip level is limited.

Another problem is the reliance on another entity to etch our circuit for us. At a bare minimum this would involve payments for the etching and shipping delays for the transport. This problem is magnified if we decide to make design changes, as every design change would require another etching, another payment, and another shipping delay. As such, this method would work best as a final implementation method after the bugs have been worked out in some other format. It is not an ideal choice for a proof-of-concept prototype.

3.2.3 FPGA Router

This implementation option was suggested to us by a number of people in the preliminary stage of the project. Initially, it was considered an unlikely option for two reasons. First, Estabrooks and Poirier indicated in their report that an FPGA implementation of the Cartesian router was not feasible because FPGA units are too expensive for this project. Second, it was believed that learning how to implement an FPGA device would be a lengthy process. FPGA's are typically programmed in Verilog or VHDL using a complex development environment. We would have to become familiar with one of the above languages, and learn how to use the development environment.

It was found, however, that FPGA's were not so expensive after all and that the Electrical Engineering department had recently acquired some FPGA development boards for a class, ECED 4260 IC Design and Fabrication. Dr. Jason Gu, the instructor for this class, was kind enough to grant us access to an FPGA board for the development of a

prototype. Furthermore, Dr. Gu also provided learning resources for MaxPlus II, the FPGA development software, as well as for structural VHDL programming.

The FPGA protoboard that is available to us is the Altera UP1 board. It supports development for the Altera MAX7000-series chips and the Altera FLEX10k-series chips. The MAX 7000 series chip (EPM7128S) that comes with the protoboard has 2500 gates and is suitable for simpler designs. The FLEX 10K series chip (EPF10K20) has over 20,000 gates, 1152 logic elements (LEs), and 6 embedded array blocks (EAB). Each EAB provides 2048 bits of memory. Either of the MAX or FLEX chips would be suitable for the Cartesian router as we have designed it.

Table 5: Typical costs for an FPGA chip[11]

FPGA Chip		Cost	Availability
Altera MAX 7000 Series EPM7128S	EPM7128ELC84-10	$73.28	Lead Time 36 days[1]
	EPM7128ELC84-12	$69.37	In Stock[1]
	EPM7128SQC160-10	$63.07	In Stock[1]
Altera FLEX 10k Series EPF10K20	EPF10K20RC240-3	$151.66	Lead Time 36 days[1]
	EPF10K20RC208-3	$133.64	Lead Time 36 days[1]

[1] Newark Electronics, http://www.newark.com/

3.3 Implementation of Proof-of-Concept Prototype

After completing the design and considering the implementation options we had to choose one and try it. Unfortunately, our first choice did not work out due to the sheer scale of it, so we had to switch implementation methods. In addition, we had to come up with a means of testing the prototype. The details can be found in the following sections.

3.3.1 The Breadboard / Logic Chip Attempt

After going through the proof-of-concept implementation options, we settled on the use of chips on a breadboard. This was done for two reasons. The first was to reduce the learning curve. A breadboard implementation would involve simply wiring up the circuits in the way we had drawn them, and it is a skill we have practiced in previous courses. The second was for ease of adjustment. In the event that there was a problem in the design, it would be a simple matter to pull a few chips and wires from one area and plug them into another. Thus, it would be handy for debugging purposes.

There turned out to be a significant problem with this approach, that being one of sheer scale. A collector router would require approximately 250 parts. An average arterial router equipped with north, south, east, and west ports would require approximately 500 parts. It should be noted that this is a rough estimate based on a count of the parts appearing in the design diagrams. AND gate chips that have four gates to a chip could decrease the actual gate count, and having to implement any of the more specialized parts (FIFOs, shift registers, counters, or MUXes) would greatly increase the count.

To further appreciate the complexity of the circuit, consider that each part would require a bare minimum of 5 wires (Vcc, ground, two inputs, and one output), requiring at least

1250 wires for a collector router and 2500 wires for an average arterial router. The actual wire count would undoubtedly be much higher, perhaps even by multiples. Any single bad wire would result in a malfunctioning router.

If we now assume that we could squeeze somewhere between 4 and 7 chips on the average department breadboard, this would have required between 35 and 65 breadboards for a collector router, and between 70 and 125 breadboards for an arterial router. By comparison, project presentations indicate that other projects this term involving breadboard circuits have fit on 1 to 4 breadboards.

Now, to get an idea of the sheer size of such a router consider that each breadboard is 16 cm by 5.5 cm, and thus has a surface area of 88 cm^2. Thus, a collector router built with chips would have a surface area between 3100 cm^2 and 5700 cm^2, and an arterial router would have a surface area between 6200 cm^2 and 11000 cm^2. So, a chip router could fit on a table surface, but getting an arterial router through a doorway without dislodging one of the 2500 wires would require at least two people and would still be awkward.

3.3.2 The Switch to FPGAs

In any case, after attempting to construct a few modules it became obvious that we would not be able to construct the prototype on time. As such, we decided to switch to using FPGAs after being provided with the lab handouts for this year's IC Design and Implementation class. The big advantage of using the FPGA is that, while we would have to learn to use the hardware description languages used to program it, we would only have to program each module once. These modules could then be downloaded into as many FPGAs as needed, thus cutting production time significantly. The FPGA chips are also flash programmable, so any design changes can easily be added to the FPGA chip simply by uploading a fresh version of the router code.

Unfortunately, the switch came too late and at the time of this writing the FPGA router is not yet complete. We hope to construct a working FPGA collector router between the end of exams and the time of the presentation, but due to inexperience with both the FPGA and its various hardware description languages we cannot estimate how successful we may be.

3.3.3 Testing the Prototype

In the event that we do succeed in constructing the router we will need a means to demonstrate that it is functioning. We will do this by writing a LabView program that can send and receive our Cartesian Router packets. Wires will be connected from the LabView connector to each of the router's ports. The LabView program will be used to transmit a packet to the router and then monitor to see if that packet was sent out the proper port. In this way we will be able to test if the router works or not.

Before getting to this final testing stage, however, a number of tests will be conducted to make sure that the individual modules work. We can use LabView to set up tests for all

of the individual modules. For example, LabView can send a packet through the PDM to see if it can successfully detect a packet. The output would be routed back to LabView and we would be able to see the results on the screen. Similar tests could be used to test the other modules.

3.4 Recommendations

Since the project has not been completed at the time of this writing, we have a number of recommendations as to what more can be done.

3.4.1 Adding State Information

The current design does not handle Cartesian Router state information. It does not send the control packets that help determine the network topology, nor does it have a means to use the information if it were received. The control packets were skipped in the interest of simplicity, to be added in a later version once a working router was produced. Once someone gets a router to work this state information should be added.

3.4.2 Adding an M-Ary Digital Communications Port[4]

If this Cartesian Router design is to ever see any sort of practical use, a better port will have to be designed than single wires carrying single bits. Single wires carrying single bits will have their speed severely limited by the available bandwidth. As bandwidth is expensive in the real world we will want to keep the bandwidth used as low as possible.

One way to decrease the bandwidth required would be to add M-Ary transmitters and receivers to the ports. M-Ary transmitters send symbols rather than individual bits down the line. These symbols consist of linear combinations of two orthogonal base waveforms, often sine and cosine waveforms. Different symbols will consist of different combinations of the base waveforms, and thus occupy different spots in the vector space defined by the base vectors. An M-Ary transmitter has M different symbols that it transmits and receives.

Each different symbol is assigned a number of bits that is dependent on the total number of symbols. In fact, each symbol will represent $k = \log_2 M$ bits. In transmission mode, the M-Ary transmitter determines which symbol corresponds to the bit combination received from our port and then sends that symbol. In receiving mode, the M-Ary receiver determines which symbol was received and what bit combination was assigned to that symbol. This bit combination can then be transmitted out the back end to the port (where we can use as much bandwidth as our hardware will allow). Thus, an M-ary transmitter will decrease our external bandwidth use by a factor of $\log_2 M$ for any given transmission speed.

3.4.3 Implementing 3-D Routing

If Cartesian Routing is ever to be used in an office building a 3rd dimension should be added to the Cartesian Routing addressing scheme. Since it is virtually inevitable that two computers on different floors will eventually have the same latitude and longitude a 3rd dimension description would be needed. Even if two machines did not share the same latitude and longitude, a third dimension would help shorten path lengths. No longer would packets have to jump from, say, the 2nd floor to the 15th floor to the 10th floor to the 20th floor while moving from east to west. This would require a small adjustment to the DMM, and only 8 bits of the overall address if the vertical dimension is entered in terms of floors.

3.4.4 Implementation as an Optical Router

It might be possible to adapt our design for use with optical logic. At the very least, the graduate student working on the optical router might find something helpful in our report, so it should be passed on to him or her (as the case may be).

In any case, the speed advantages provided by an optical Cartesian Router in an optical network make this option well worth pursuing.

4.0 Conclusion

The goal of this project was to design and build a Cartesian router using a hardware-based decision engine. Unfortunately, at this time, we must report that we have a router design, but that we do not have a completed router prototype.

We attribute our inability to provide a prototype on schedule primarily to our initial decision to implement the router using discrete TTL and CMOS components. After attempting to implement a few of the modules, we realized that there was little chance of finishing the prototype in time. When we reconsidered our options and discussed our situation, we decided that switching to an FPGA implementation was the ideal choice.

Because of the 11th hour decision to switch our implementation method, the FPGA-based prototype is not completed at this time. However, with the guidance of Dr. Hughes and with the resources provided by Dr. Gu, we hope to have a working FPGA collector router by the time of the project presentation. In addition, we plan to conduct tests in order to illustrate that the performance of the router is up to the standards requested by Dr. Hughes.

References

Papers and Reports

[1] Kevin Estabrooks and Pascal Poirier, "Simplified Cartesian Routing", Dalhousie University, December 2000.

[2] Larry Hughes, Omid Banyasad, and Evan Hughes, "Cartesian Routing", Computer Networks, Elsevier, April 2000.

[3] Larry Hughes and Omid Banyasad, "Cartesian Routing: Progress to Date and Future Directions", Workshop on New Visions for Large-Scale Networks: Research and Applications, February 2001.

[4] Jacek Ilow, "ECED4503 Digital Communications Systems Slide Notes", Dalhousie University, January 2001.

Books

[5] Clayton L. Hallmark, "The Master IC Cookbook", Tab Books, Inc., Blue Ridge Summit, PA, 1981, pp. 9-473.

[6] L. Hughes, "An Introduction to Computer Communications", Whale Lake Press, Halifax, NS, 2001, pp. 3-18, 23-39, 117-128, 211-225.

[7] Randy H. Katz, "Contemporary Logic Design", The Benjamin/Cummings Publishing Company, Inc, Redwood City, CA, 1994, pp.1-100, 122-151, 173-207, 283-298, 329-337, 356-364, 402, and 517.

Websites

[8] http://www.altera.com/education/univ/unv-kits.html, The Altera University Program: Design Laboratory Kits Website.

[9] http://www.dal.ca/~jgu/index.html, Dr. Jason Gu's Website.

[10] http://www.fairchildsemi.com, Fairchild Semiconductor Website

[11] http://www.findchips.com, FindChips.com Website

[12] http://www.dal.ca/~lhughes2/cartnet/index.html, Cartesian Routing, Dr. Hughes' Website