**ABSTRACT**

PHELPS, BRIAN ROBERSON. Hardware realization and implementation issues for the Sliding-Window Packet Switch (Under the direction of Dr. Paul D. Franzon)

Shared memory packet switches are known to provide the best delay-throughput and respond well to bursty traffic. Shared memory switches are also known to scale poorly due to centralized control and memory bottlenecks. The Sliding Window Packet Switch (SW) algorithm is a shared memory switch that employs decentralized control and multiple memory modules to facilitate the scalability of hardware. The SW algorithm is independent of the type of packet or cell.

This research has two closely related goals. The first goal is to implement the SW algorithm in hardware such as an FPGA. This implementation is actually a specific case of the SW algorithm with four input ports and four output ports (i.e. a 4x4 switch). The second goal is to determine what scalability constraints exist in hardware for larger numbers of input and output ports (large NxN). These constraints are used to predict the overall throughput that the hardware implementation can handle.

# HARDWARE REALIZATION AND IMPLEMENTATION ISSUES FOR THE SLIDING-WINDOW PACKET SWITCH

**by**
**BRIAN ROBERSON PHELPS**

A thesis submitted to the Graduate Faculty of
North Carolina State University in partial
fulfillment of the requirements for the Degree of
Master of Science

**COMPUTER ENGINEERING**

Raleigh

2004

**APPROVED BY:**

| | |
|---|---|
| Dr. Paul D. Franzon<br>Committee Chair | Dr. Sanjeev Kumar<br>External Member |
| Dr. Thomas M. Conte | Dr. Arne A. Nilsson |

# DEDICATION

I dedicate this to my family. Without the support of my parents Bob and Pat or my fiancé Kaitlan, I would have never attended graduate school. Their tolerance during stressful times has reminded me what wonderful people they really are. My crazy dog, Zappa, has also been helpful by not chewing too many of my belongings lately.

# BIOGRAPHY

Brian Roberson Phelps was born in Durham, North Carolina on December 1st, 1977. After graduating from Southern Durham High School in 1996, he enrolled in North Carolina State University. Brian graduated in May of 2001 with a B.S. in Electrical Engineering and a B.S. in Computer Engineering. After graduating, Brian worked as a systems technician for a lighting and sound company in Durham, where he is currently employed.

# ACKNOWLEDGEMENTS

I was very lucky to have the help of so many wonderful people during this project inside and outside of this university.

Dr. Paul Franzon supplied me with a wonderful opportunity to try research and he gave me invaluable advice on this project. I could not have asked for a better topic. Dr. Sanjeev Kumar's guidance on this project made completing this project possible. His support through the many long-distance phone calls and other communications was greatly appreciated. I also appreciate Dr. Tom Conte and Dr. Arne Nilsson for their participation on my advisory committee.

I would like to thank David Winick and Steve Lipa for putting up with me in EGRC 301 and their advice and help on anything computer related. Also I thank Dr. John Wilson, who allowed me to use some of his CPU cycles to finish some of my simulations on his blazing fast Linux machine: bock.ece.ncsu.edu.

# TABLE OF CONTENTS

# LIST OF TABLES

Page

# LIST OF FIGURES

Page

# Chapter 1

# Introduction

The Sliding Window (SW) switching architecture is a new class of high performance of algorithm driven switching architecture [1]. This thesis will study the shared memory based version of this algorithm. Shared memory based switches are currently known to perform well under bursty traffic, but scale poorly for a large number of input and output ports (large NxN). Figure 1.1 shows an example of a simple switch. The scalability of conventional shared memory switches is largely limited by the following:

- Centralized control for all of the stages.
- Memory bandwidth is shared between ports.
- Memory space is segregated.

The SW algorithm avoids Centralized control by having five independent pipeline stages. These pipeline stages need only the information available locally. Having multiple memory modules parallelizes memory bandwidth. This allows the packets arriving from each port to be written simultaneously. Memory space is completely shared between all ports, so there is more room to store large bursts of traffic. By avoiding the listed limitations, the SW algorithm is better suited for scaling to larger NxN. The goal of this research is not only to implement the SW switch algorithm in hardware, but to also predict how large the SW algorithm can scale in modern hardware and what the hardware limitations will be.

To understand fully how the SW algorithm will translate into hardware, it is best to build a small working prototype, like a 4x4 switch. This is accomplished using an Altera FPGA and Quartus II software for synthesis. This model is then used to determine what the scalability bottlenecks and practicality issues would be for large NxN.

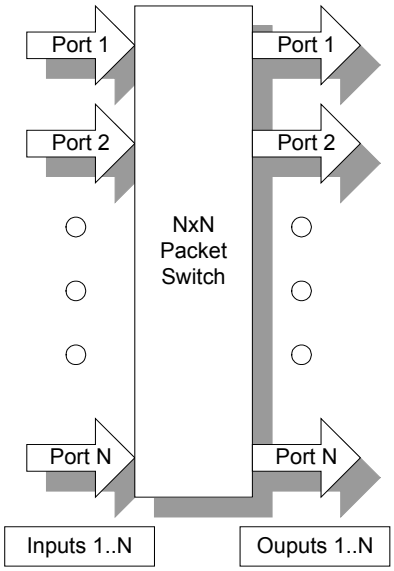



Figure 1.1 Example of a simple network Switch

## 1.1 Organization of the Thesis

Chapter 2 will briefly discuss several different network-switching algorithms and a brief comparison to SW. Chapter 3 will outline my implementation of a 4x4 SW switch in an FPGA with parameters $\sigma = 12$, $m = 8$, and $p = 2$. These parameters define the way the memory is handled in the algorithm. This thesis will not go into great detail about the algorithm. One is expected to reference Dr. Sanjeev Kumar's work in [1] on this subject for more information.

Chapter 4 will discuss the scalability issues uncovered by the hardware implementation of the 4x4 switch. In this chapter the speed of the algorithm and memory size requirements will be discussed as well as how these issues limit the number of ports. Chapter 5 will

provide verification results on the hardware implementation of this 4x4 SW switch. Chapter 6 will provide conclusions that the research uncovered and what the future plans are for the implementation of this algorithm in hardware.

# Chapter 2

# Other Network Switch Architectures

Modern shared-memory switches such as presented in [2] employ centralized control. They also must share memory bandwidth between ports. As port quantity increases, the tasks that the controller must accomplish also increases. In addition, the memory bandwidth that is shared between ports becomes a problem. These factors limit the scalability of this switch architecture.

Some of these shared-memory switch architectures such as the one presented in [3] distribute bandwidth among multiple memory modules. This architecture also employs a centralized controller for all functions. As port quantity grows, so does the required memory and therefore the length of the search through shared memory space. This architecture also requires that the controller manage all stages of the switch. Read and write operations, searches, and other functions grow as switch size grows. For these reasons, this architecture is known to have limits on scalability.

To counteract this scalability problem, many smaller shared memory switches are connected together to achieve a larger port quantity [4] such as in Figure 2.1. In this figure, four 2x2 shared memory switches are connected together to achieve a 4x4 switch. This is known as a Multistage Interconnection Network (MIN) [5].

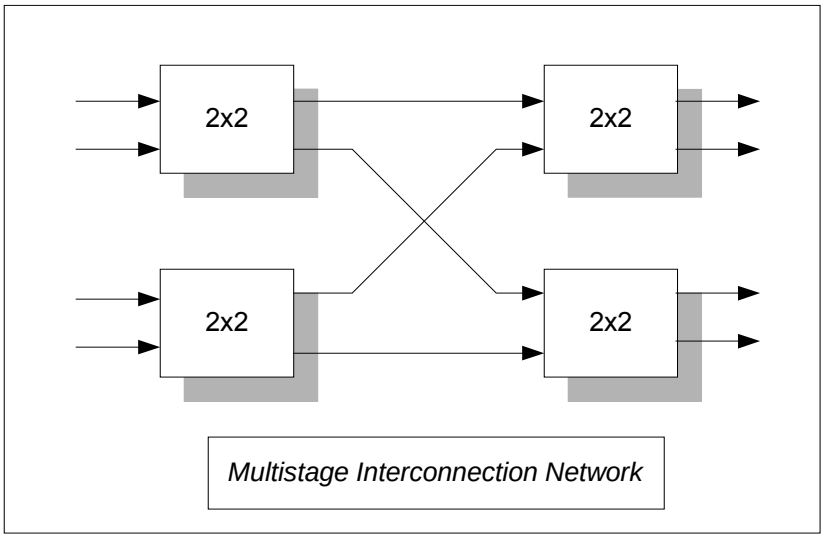



Figure 2.1 Example of a MIN

The hardware for these shared memory switch architectures is known not to scale much larger than 16x16. Although a larger port quantity is achievable using MIN as mentioned above, these packet switches are known to have poor throughput for high network traffic loads. As an example, consider connecting multiple 8x8 shared memory packet switches in MIN fashion to obtain a 64x64 packet switch. The 8x8 switch itself has a nearly linear relation of traffic load versus throughput (Figure 2.2 from [1]). However, when it is grown as a MIN to 64x64, the throughput never breaks 0.6 (Figure 2.3 from [1]).

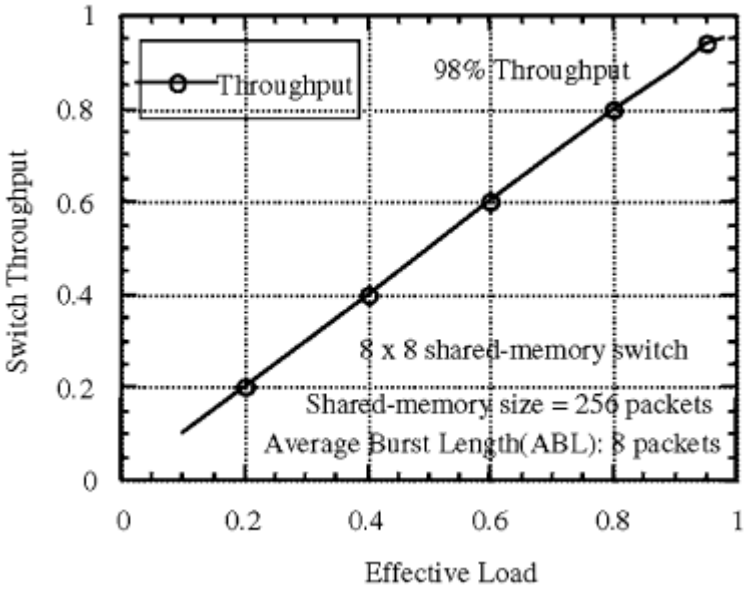



Figure 2.2 Nearly linear load is achieved with an individual 8x8 switch (from [1])

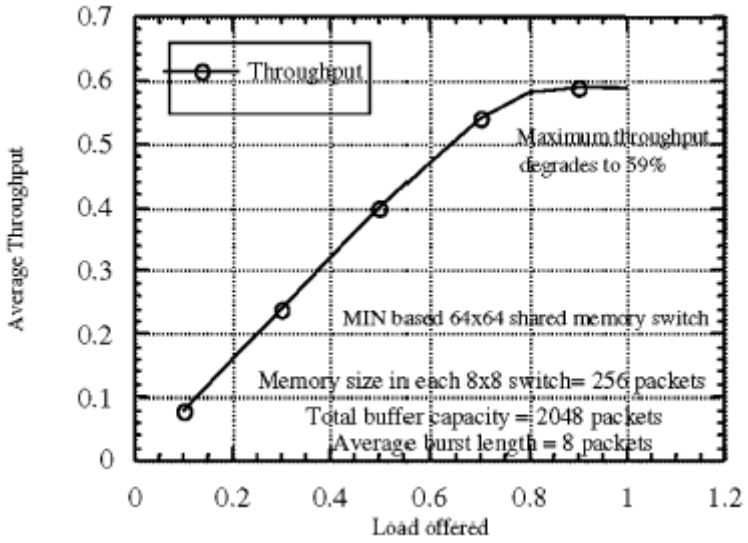



Figure 2.3 64x64 MIN based on the 8x8 shared memory switch (from [1])

## 2.1 A Comparison of Architectures

To achieve high throughput for switches with large port quantities, a new switching architecture must be designed. The Sliding-Window (SW) Switching Architecture avoids the limiting factors of the other architectures by using decentralized control and shared

memory modules. As a comparison, consider a 64x64 SW switch with the same buffer capacity shown in Figure 2.4 from [1].

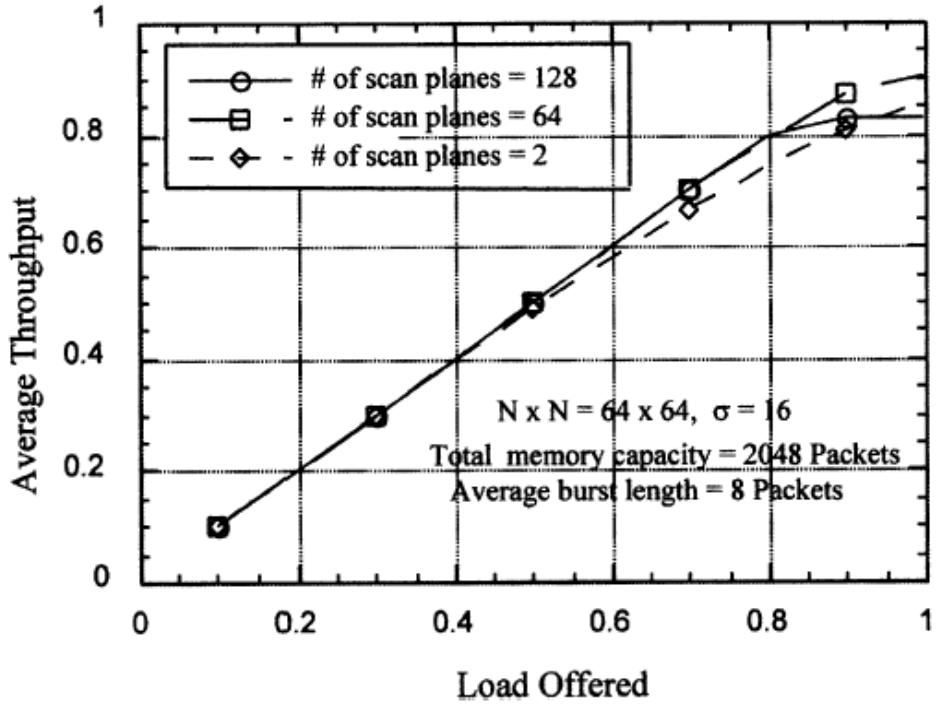



Figure 2.4 Load versus throughput for a SW switch (from [1])

For the same traffic and buffer memory, the SW architecture offers near-linear load versus throughput. A throughput of almost 0.9 is achievable with this architecture. MIN plateaus before 0.6 with traffic and memory being equal. SW offers significant improvements over conventional MIN based switches.

# Chapter 3

# Implementation

This thesis will not provide a review of the Sliding-Window (SW) switching algorithm. An excellent explanation of this algorithm may be found in [1]. This chapter will often reference this article to explain the architecture. The SW algorithm is a generic switching algorithm that is independent of the type of packet. The overall function of this algorithm is to queue and route a given packet to the specified output port.

## 3.1 Method

A software version of this algorithm was created in Matlab for debugging and verification purposes. Matlab was chosen because of its excellent matrixing capabilities. Performance of the Matlab comparison algorithm is also not really an issue since the HDL version will be much slower.

The overall structure of the algorithm, and the hardware, is shown in Figure 3.1. Notice there are five distinct pipeline stages to the entire algorithm:

1. The Parameter Assignment Circuit (PAC)
2. The Input Interconnection Network (IIN)
3. The Read Operation (Memory Module input)
4. The Write Operation (Memory Module output)
5. The Output Interconnection Network (OIN)

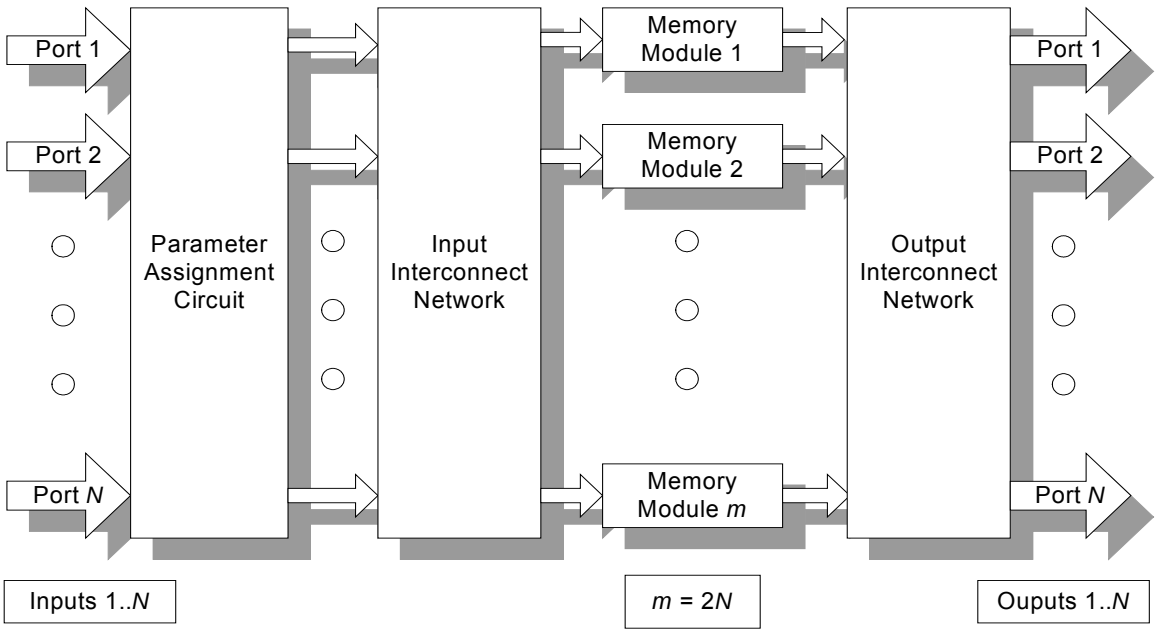



Figure 3.1 Architecture of the Sliding-Window Switching Algorithm

Before the hardware can be designed, a certain set of assumptions must be founded. They are as follows:

1. Design a 4 input, 4 output switch ($N = 4$)
2. The Packets are 64 bytes long.
3. The header ($d$) is in the first few bits.
4. $\sigma = 12$, $m = 8$, $p = 2$.
5. The data arrives serially and leaves serially.
6. The target hardware is the Altera EP20K400EFC672-3

The packet length of 64 bytes is the size of a standard ATM cell. The header is assumed to contain $d$ and is located in the first byte of the packet. Dr. Sanjeev Kumar suggested that the parameters $\sigma = 12$, $m = 8$, and $p = 2$ be used for this design. For testing purposes the data is assumed to arrive serially. This is because the NI Data Acquisition Card

has only 24 channels. Serial data for each port would require 4 inputs + 4 outputs + 1 clock + 1 reset = 10 channels. This is not including any debugging signals.

The Altera FPGA has certain limitations that must be considered. It has about 16,000 registers, but 239,000 bits of configurable RAM. This is important considering the packets are 64 bytes (512 bits). The number of packets that must be stored are determined by the parameters $\sigma = 12$ and $m = 8$ to yield:

$$8 \times 12 \times 512 = 49{,}152 bits$$

This large number of bits is too large to fit in the registers, but small enough to fit in the RAM. To use the RAM and Registers efficiently, and to design hardware efficiently in general, block diagrams of the hardware must be created before coding the HDL models.

The Altera FPGA must be programmed from a binary file created by the Quartus II software. This software synthesizes the HDL to a binary that may be downloaded to the FPGA. Quartus II supports several HDL's, but I chose Verilog to implement this hardware because of my familiarity with this language.

## 3.2 Parameter Assignment Circuit Design

The most challenging part of the entire SW algorithm to design in hardware is the Parameter Assignment Circuit (PAC), which is the first stage of the switch. It contains counters, variables, and tables that must be analyzed and updated accordingly. The PAC assigns the self-routing parameters *i, j, k* to each packet. These parameters allow the packet to flow through the rest of the switch stages on its own, without any centralized control between the switch stages.

The core structure of the PAC hardware is based on the Processor Core diagram, which is shown here in figure 3.2. This diagram is based on Figure 6 of [1]. Note that the

PAC core is broken down into two distinct sub-pipeline stages of processors (Processor 1 and Processor 2). This is because the entire PAC, and therefore the pipeline, operates sequentially on a given set of packets during a given switch cycle. The parameters assigned to a packet have a direct dependency on previous packet assignments. This dependency would be difficult, if not impractical to predict. This makes it extremely difficult to pipeline or parallelize this parameter assignment hardware much further. The rest of the PAC is designed around this Processor Core.

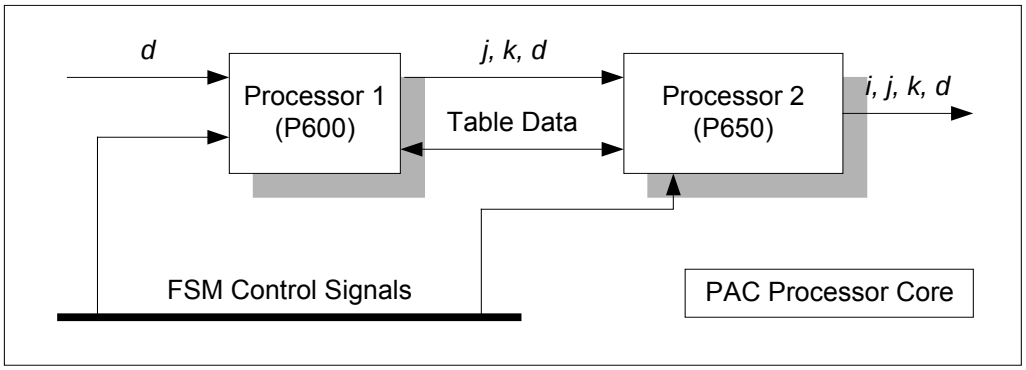



Figure 3.2 Parameter Assignment Circuit Core

P600 assigns the *j* and *k* parameters while P650 uses *j* and *k* to compute the *i* parameter. Both processors work together to handle queue control, although P600 does most of this. The overall function of the PAC is read a packet from an input port, assign the parameters, and repeat until all the packets on the input have been assigned. This process is defined as one switch-cycle. The PAC must store all the packets from the input each switch-cycle as they arrive because it is a pipeline stage.

### 3.2.1 PAC Header Circuit design

It would be convenient to grab the *d* parameter out of the header as we parallelize the first group of bits. The *PAC_Header.v* module not only parallelizes the serial data, it also

extracts *d* at the earliest possible time. The FSM module, *PAC_NCFSM.v*, triggers this capturing of *d*. The data parallelizing is done using a shift register macro created by the Quartus II Mega-Function Wizard. This macro just uses registers, and not RAM, since there is no way to bitwise shift the on chip RAM. It is created in the same way as the RAM is using the Quartus II *Project Megafunction Wizard*. The shift register converts the 512 bits of serial data to 8 x 64 bit words.

A block diagram of the hardware to create this module is shown in Fig. 3.3. There are four of these modules in the upper-level design *PAC_Full.v* corresponding to each input port. All the inputs and outputs of the modules are labeled with <signal>_in or <signal>_out. In addition, <signal>_in will be connected to <signal>_out of the previous module. This is done for clarity of signal flow. Another mantra used in this design is that algorithm variables such as *d* are actually one extra bit longer than necessary. The reason for this is that all algorithm counters and variables start at one. This allows the hardware to follow the algorithm, which also starts at one, to avoid confusion. It also lets zero be the special case of having no packet. It will be shown later that this is crucial for the hardware to work properly.

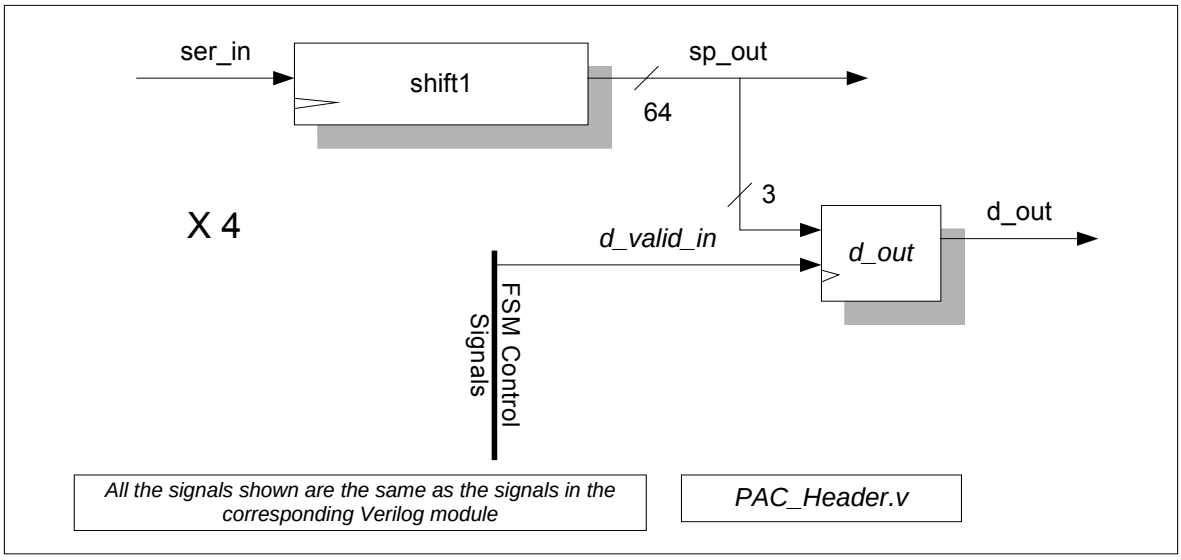



Figure 3.3 The Header Module

### 3.2.2 PAC Memory design

Recall that PAC is a switch pipeline stage. This implies a one-packet delay each switch cycle that must be implemented with some type of memory. So the PAC must store the packet somewhere each cycle, preferably in RAM, otherwise 512 bits x 4 inputs = 2048 registers will be wasted. This RAM is created using the Quartus II *Project Megafunction Wizard* just like the shift register in the header. This packet delay is handled by reading the current write address plus one. Therefore, each memory writes (and reads) the packet in 8 cycles per switch cycle. The read address will read what is currently being written 8 memory cycles (one switch cycle) later.

*PAC_Memory.v* also handles injecting the newly assigned *i, j, k* parameters (from the P650) back into the packet. Once the parameters are found, they are temporarily stored until they can be injected into the packet at the right time using mux1 and the *ctrl1_in* signal from the FSM. All control signals and addresses are handled by the FSM. The block diagram for the *PAC_Memory.v* module is shown in Figure 34. Since there are four input ports there

must be four of these connected to the *sp_out* output of each Header module. Please note that the PAC memory module is not related to the Memory Modules involved in queuing in the third and fourth pipeline stages.

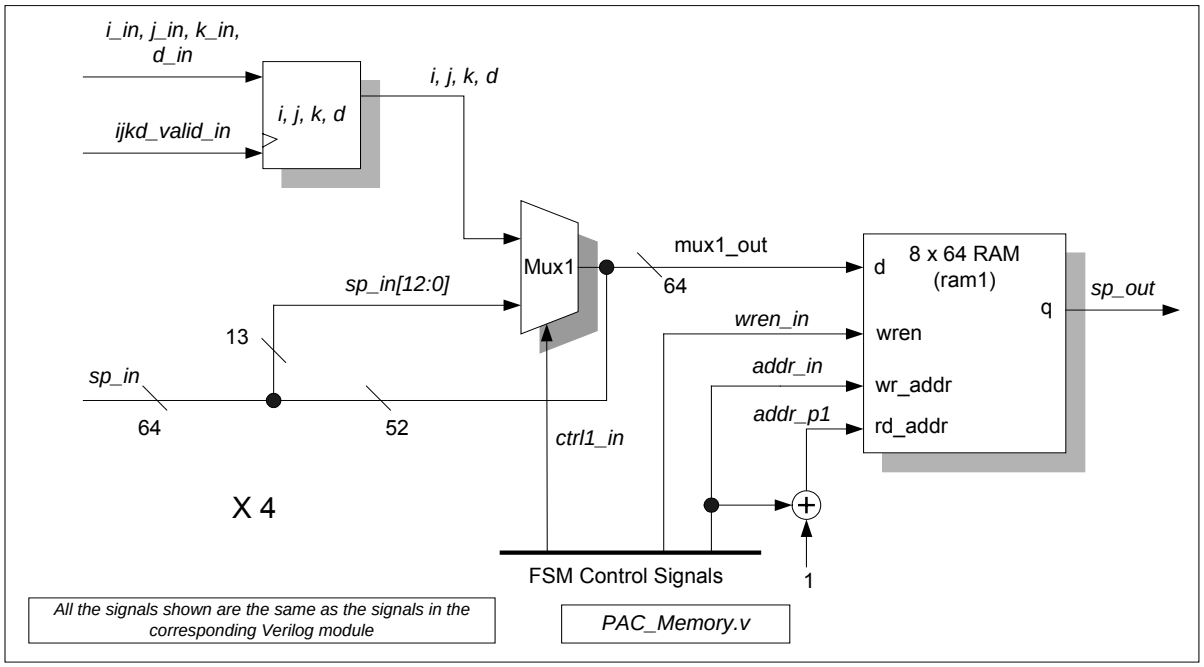



Figure 3.4 The PAC Memory Module

### 3.2.3 The PAC Finite State Machine

The PAC control signals are generated by one Finite State Machine (FSM), the *PAC_NCFSM.v* module. This module is simply a 9-bit counter that sends out control signals based on the current counter values. The simplest way to accomplish control for many events that are strictly time dependent, such as in the PAC, is just to use a counter and logic. This is not a typical FSM because it has no inputs other than *clock* and a *reset*. However, according to the definition, it is actually a Moore FSM because the output only depends on the current state (counter). The block diagram for this hardware is shown in Figure 3.5.

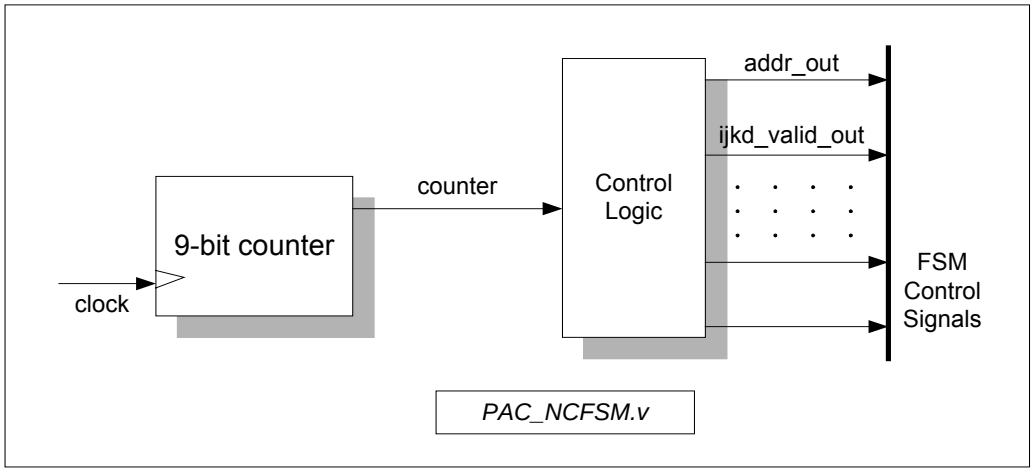



Figure 3.5 The PAC Finite State Machine

### 3.2.4 PAC SW-Counter

Each processor has its own Sliding-Window Counter. This counter updates according to the SW algorithm given in [1]. This counter is a two-digit modular counter that updates when signaled by the FSM. The first digit is SW_j and increments mod $\sigma$. The second digit, *SW_k,* increments mod *p* only when SW_j == 1. Recall that all algorithm variables such as the Sliding-Window Counters *SW_k* and *SW_j* start at (reset to) one instead of zero. This SW counter is what the *j* and *k* values are based on. It also determines when portions of the tables are updated in both processors such as the Scan Table (*ST*) in P650. A block diagram of this counter is shown in Figure 3.2.5. This diagram corresponds to the Verilog module *PAC_SWCounter.v*.

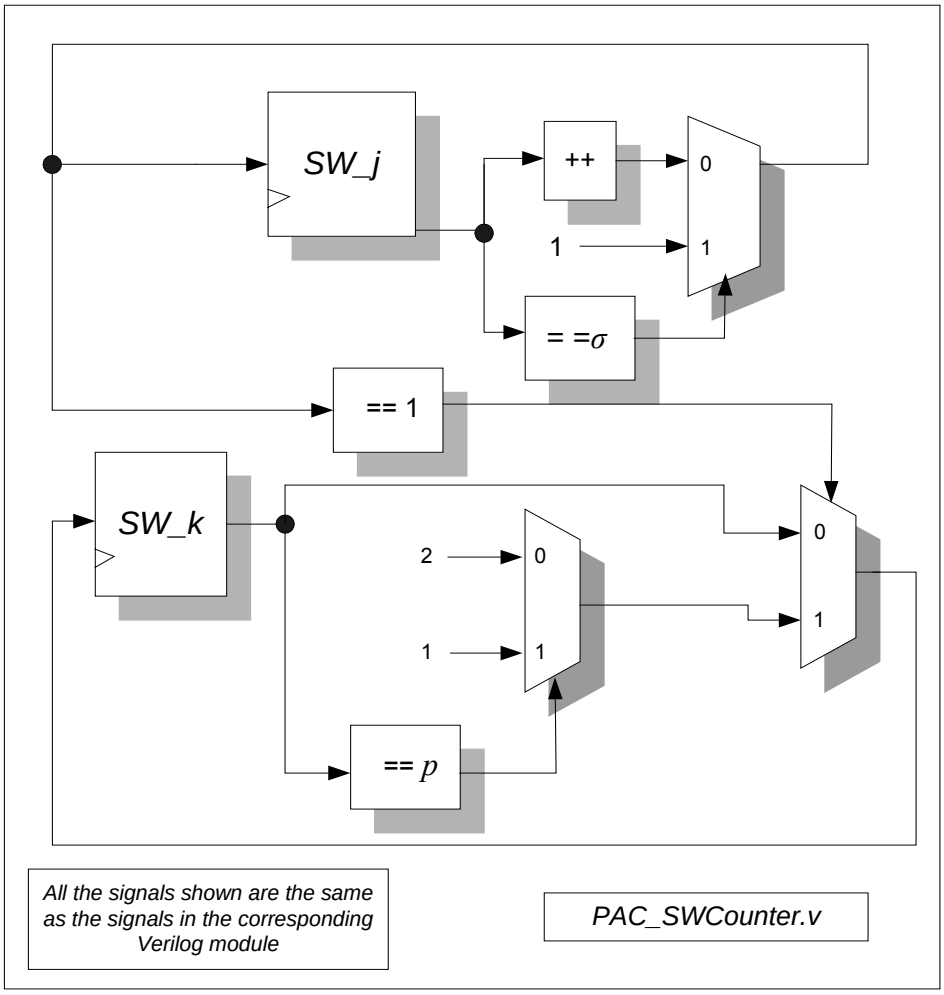



Figure 3.6 The Sliding Window Counter

### 3.2.5 PAC Processor 1

The first stage of the PAC core is P600. This processor uses counters, tables, and conditions to assign the parameters *j* and *k*. The design is relatively straightforward and based on the flowchart given in Figure 3.7 from [1]. Most of this flowchart translates straight into hardware easily.

Tables such as Qd[d], LC_j[d], and LC_k[d] are simply implemented in register based RAM, not pre-compiled on chip RAM like memory intensive packet storage. LC_j[d]

and LC_k[d] are used to keep track of the previous *j and k* parameter assignments to a particular output port *d*. Qd[d] is a queue counter that holds the number of packets destined to the corresponding output port. All of the conditional dependencies for P600 allow the entire *j, k* assignments and queue control to be easily completed in one clock (or packet) cycle per packet. After all the packets have been assigned there is an update cycle triggered by the FSM's *SW_enable* signal. This update cycle increments the *SW_j* and *SW_k* counters according to the SW scheme. It also simultaneously decrements the *Qd[d]* arrays that contain the queue length for each port, because each port always removes a packet from the queue every switch cycle.

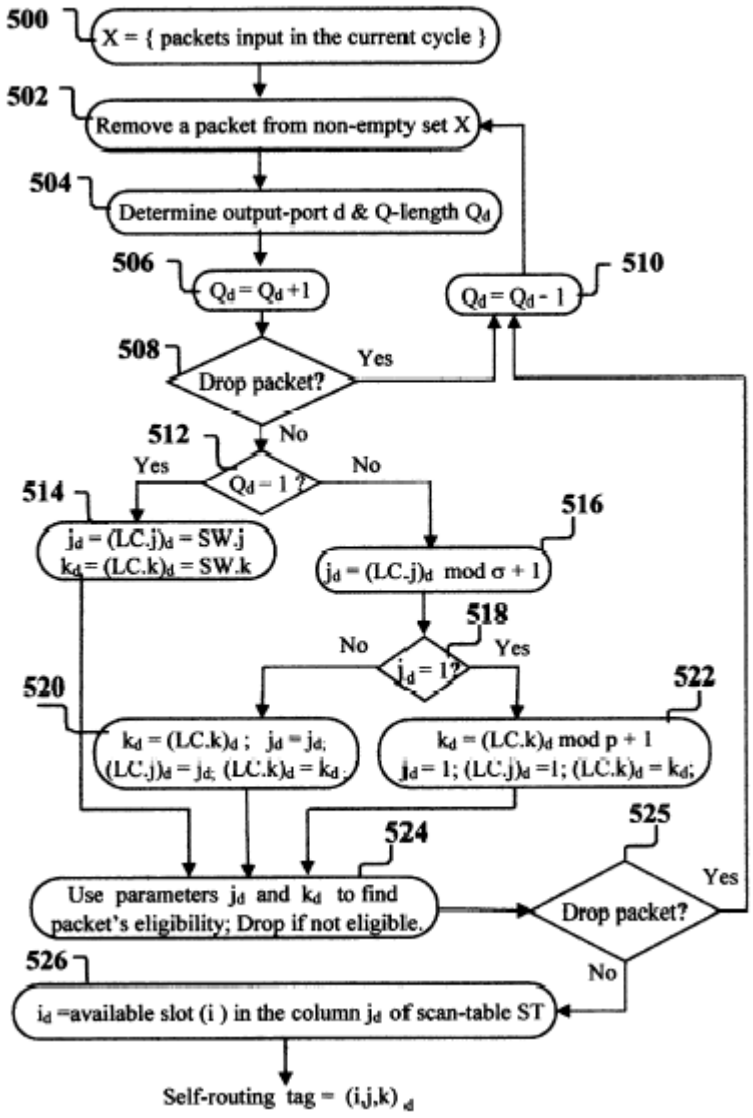



Figure 3.7 Flowchart of P600 operations (Figure 5 from [1])

Step 524 is completed using the following pseudo code:

```
if (Qd > p. σ)
   drop the packet
   Qd = Qd - 1
else if (Qd > σ)&&(Qd <= p. σ)
   if (ES_j <= r)
      drop the packet
      Qd = Qd - 1
```

*ES_j* is a scalar that contains the number of empty slots in vector *ST[j]* and *r* is the number of input ports that have queue lengths less than $\sigma$.

There is unfortunately a catch in the algorithm to hardware translation that makes one small part, the *ES_j* calculation, not so straightforward. *ES_j* depends on the *j_524* vector of the Scan Table: *ST[j_524]*. Since *ST* is in P650, *ES_j* is calculated by P650 based on the current *j_524* value from P600. This *ES_j* is then returned to P600, where P600 uses this to decide whether to drop the packet. The reason this must be looped backwards in the pipeline is because of the data dependency. This *ES_j* calculation depends on *ST[j_524]* which is only in P650. In addition, P650 depends on the current *j* and *k* to update the *ST* in the next packet cycle. However, *j* and *k* are not valid (i.e. they may be dropped) until *ES_j* is calculated and compared in P600. So on top of having to loop this *ES_j* value backwards in the pipeline from P650 to P600 we must also speculate when calculating *ES_j* what *ST[j_524]* will be after P650 updates with the current *j, k values*. Please note that *j* is the same as *j_524* delayed by a cycle (and dropped if it fails 524 or 508). Also, j_524 affects no other decisions or registers in P650 directly in this calculation. The *ES_j* calculation is essentially an isolated circuit in P650 that reads the Scan Table. The hardware block diagram of P600 is displayed in Figure 3.8.

Note that there is only one P600 per switch. The reason this is so is because the algorithm depends heavily on the previous packet's parameters. This dependency makes it difficult to assign these parameters any other way than sequentially. Since the assignment is sequential, only one processor is needed per switch to assign *j* and *k*. The same is true for P650.

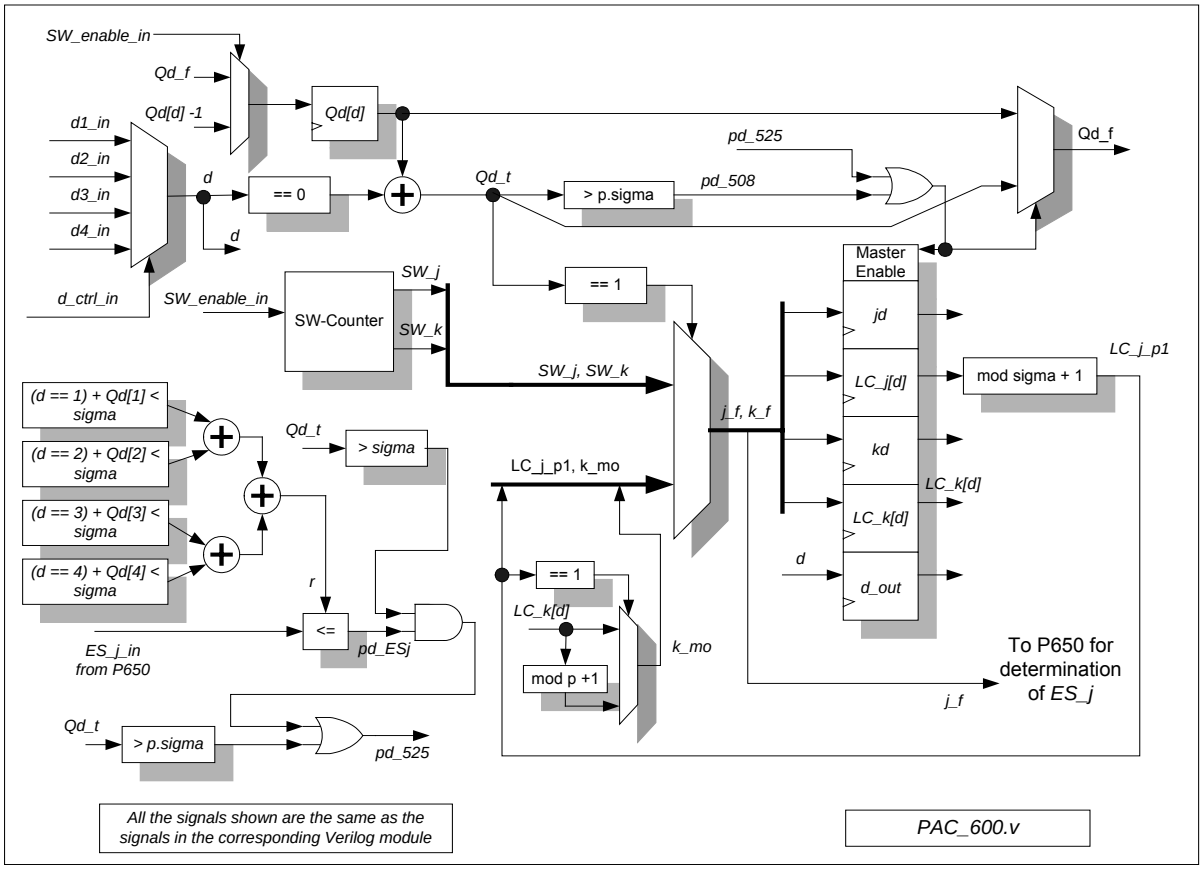



Figure 3.8 P600 design

### 3.2.6 PAC Processor 2

The second stage of the Processor pipeline is P650. This processor's main duty is to assign the *i* parameter. It also maintains the Scan Table *ST* and calculates *ES_j* for P600 based on the current j_524 value. *ST* is a two-dimensional table (*j*, *i*) that holds the *k* values for the previously assigned packets. This table's purpose is to make sure P650 writes the packet to an empty memory slot.

*ST* is updated to reflect reads at the end of the switch cycle after all the packets have been assigned. This update happens simultaneously with the update of the SW-Counters (just as P600 updates *Qd[d]* and its SW-Counters simultaneously) . The *ST* update consists

of clearing all the *k*'s stored in vector *ST[SW_j_addr]* that are the same as the *SW_k* value. This corresponds to reading out a packet from the memory to the output port. This *SW_j, SW_k* value matches the three-switch cycle delayed value of the counter in the memory Read-Out operation (discussed later). The block diagram of the update hardware for P650 is shown in Figure 3.9. Recall that all algorithm variables such as *SW_j* and *SW_k* start at one and hardware such as RAM starts at zero. To bridge this gap the equivalent hardware address is calculated by subtracting one from the SW algorithm variable. The resulting signal is <signal>_addr. An example is: *SW_j_addr* = *SW_j* – 1;

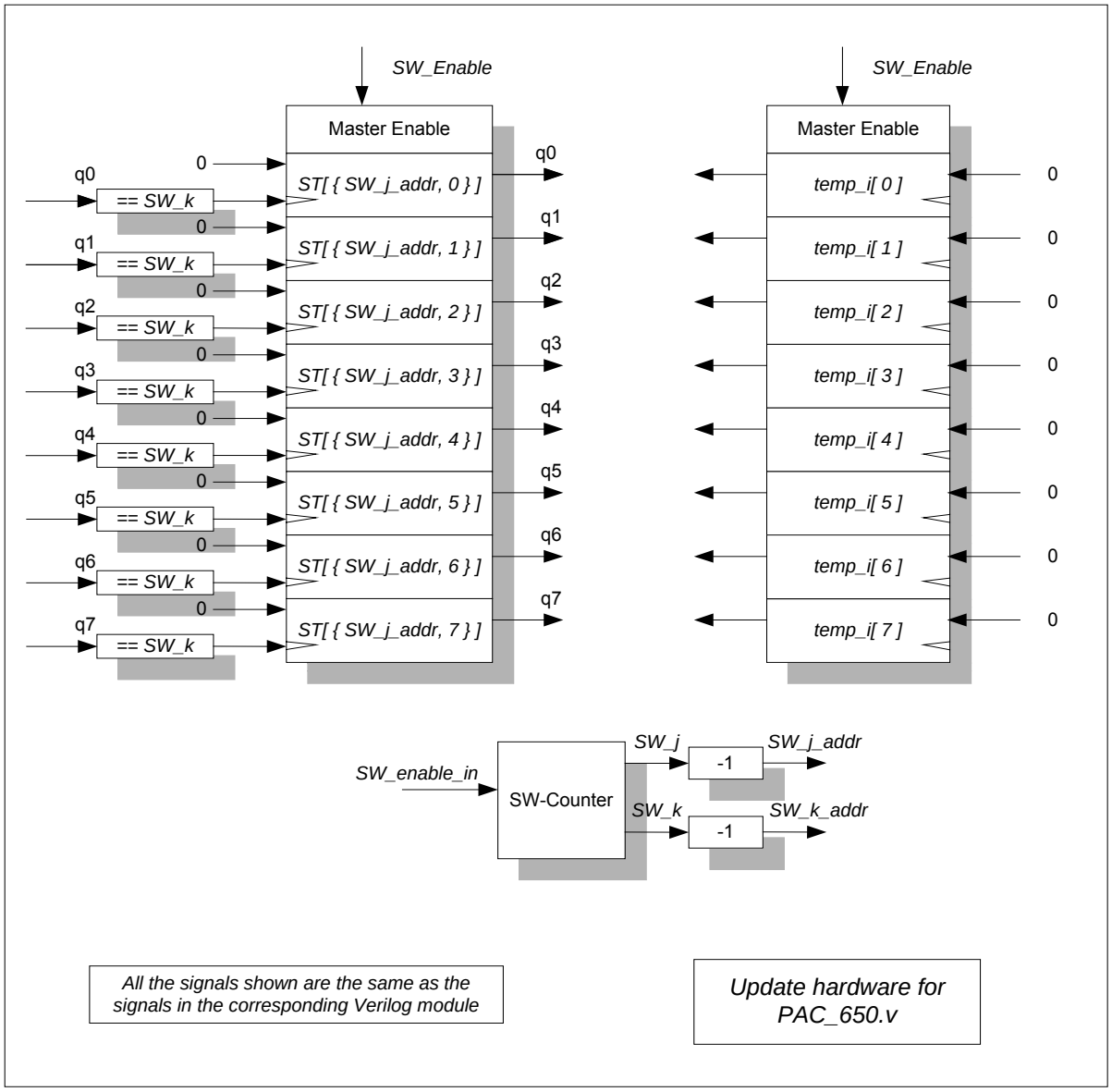



Figure 3.9 Update Hardware for P650

P650 must find an empty slot in the *ST[j_addr]* vector, and assign the parameter *i* to this location. There is also another algorithm to hardware translation problem here. Each switch-cycle all the packets must simultaneously be routed from the PAC to the Memory Module denoted by *i*. This routing is very difficult to do in hardware if two packets have the same *i* value in one switch cycle. This duplicate *i* value is possible because the *j* can be different for two packets while *i* is the same. The memory bus speed on the Memory Module

must also be doubled for this purpose of writing two packets simultaneously. The good news is that the larger the switch is (in port quantity), the less likely this is to happen. Also the chance of more than two packets requiring the same *i* value for a 4x4 switch is less than one in ten million [11].

To avoid a duplicate *i* value, a register *temp_i* is used to hold the previous *i* assignments for the current switch cycle. This *temp_i* register has *m* bits, or a bit for each Memory Module. This register is compared with *ST[j]* using a bitwise OR. Then this output is searched bit by bit for an open or zero slot. A search implies a priority encoder in hardware. The register is cleared at the end of every switch cycle.

The output of the bitwise OR operation mentioned above is the select-line for the priority encoder *PE1*. Note that the select line of PE1 and PE2 can be thought of as inverted, so that the first "0" slot is found instead of the first "1" slot. The inputs to the priority encoder are hard-wired numbers corresponding to each priority encoder input number (see Figure 3.10). For this switch, there are 8 select lines and 9 sets of inputs. Note that one of the inputs to the priority encoder is the number 8 and specifies the default case, or an unsuccessful search. The priority encoder finds the first empty (zero) slot (starting with the MSB) on the select line and outputs the corresponding input, which has the corresponding *i* location. Since the MSB has highest priority, *temp_i* and *ST[j_addr]* are flipped bitwise in the actual hardware implementation to match the SW algorithm. This flipping is arbitrary and will be ignored in this example:

$ST[\ j\_\text{addr}\ ] = 11101010$
$temp\ \_\ i = 10000000$
$(temp\ \_\ i)\ |\ (ST[\ j\_\text{addr}\ ]) = 11101010$
$PE1\_select\ \_\ line = 11101010$
$PE1\_inputs = 1,2,3,4,5,6,7,8$
$PE1\_output = 4$

If an *i* cannot be located in this first search, a second *i* search is performed simultaneously with *PE2,* which is another priority encoder. This search is only done on the *ST[j_addr]* vector. The result of this search is used only when the first search fails, which means this must be a duplicate *i* value. The block diagram of the *i* assignment hardware for P650 is shown in Figure 3.10.

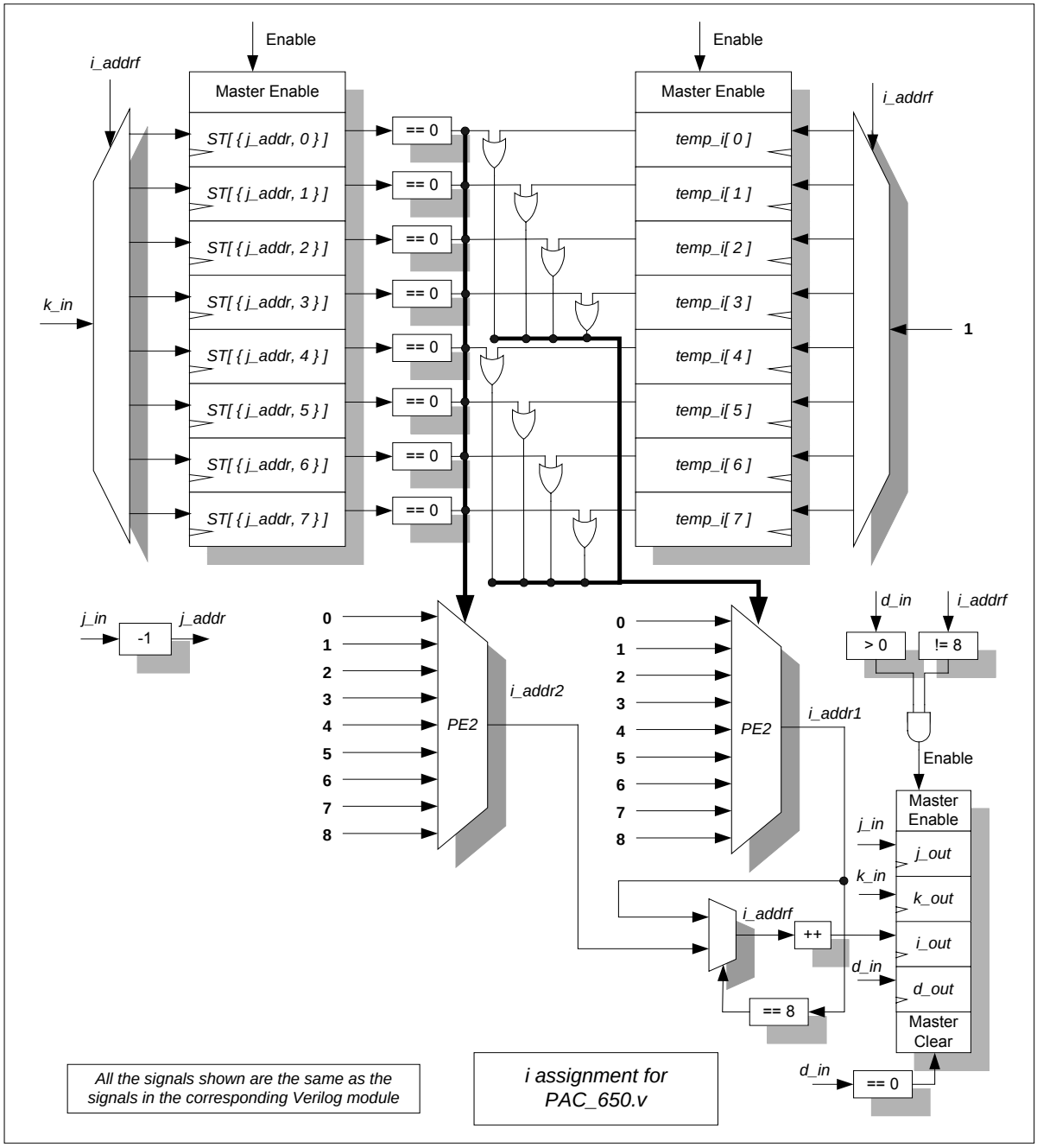



Figure 3.10 Parameter *i* assignment for P650

P650 must also use *ST* to calculate the *ES_j* value for P600 based on *j_524*. *ES_j* holds the number of empty slots in the *ST[j_524_addr]* vector (which holds the *k* values of the packets in the *j_524_addr* vector). To calculate *ES_j*, a comparator is used on each of the data locations of the vector *ST[j_524_addr]* to find empty slots. Recall that this vector holds

all the *k* values for the $j^{th}$ index of all the Memory Modules. The results of these comparators are added together to obtain *ES_j*. If the *j_524_in* is equal to the current *j_in* value then we must speculate that *ES_j* is actually *ES_j* - 1. This is because the parameters *j* and *k* on the input have not been written to the *ST* yet. If it has the same *j* as the *j_524* (the future packet) then we must speculate that this packet occupies a slot in the *ST[j]*. This is not the case yet because of the pipeline, as stated earlier. The way to get around this problem is to subtract one from *ES_j* after the calculation. In summary P650, and therefore *ST,* can be thought of as a cycle behind P600 because of the pipelining of these two processors. P600 needs data that is effectively a cycle behind and must use informed speculation to access this data. A block diagram of this calculation hardware is shown in Figure 3.11.

*ST* is a two-dimensional table indexed by ( *j, i* ). This is implemented in hardware using register based RAM, since it is relatively small. The address to one-dimensional RAM is calculated by concatenating the *j* and *i* address bits. This effectively creates an efficient two-dimensional address space assuming *i* (the lower bits of the address) has a maximum value that is a power of 2 minus 1 such as $7 = 2^3 - 1$.

For example if j = 5 and i = 2 then the address {j_addr, i_addr} in binary is:

$\{4, 1\} = \{0100\ 001_b\}$

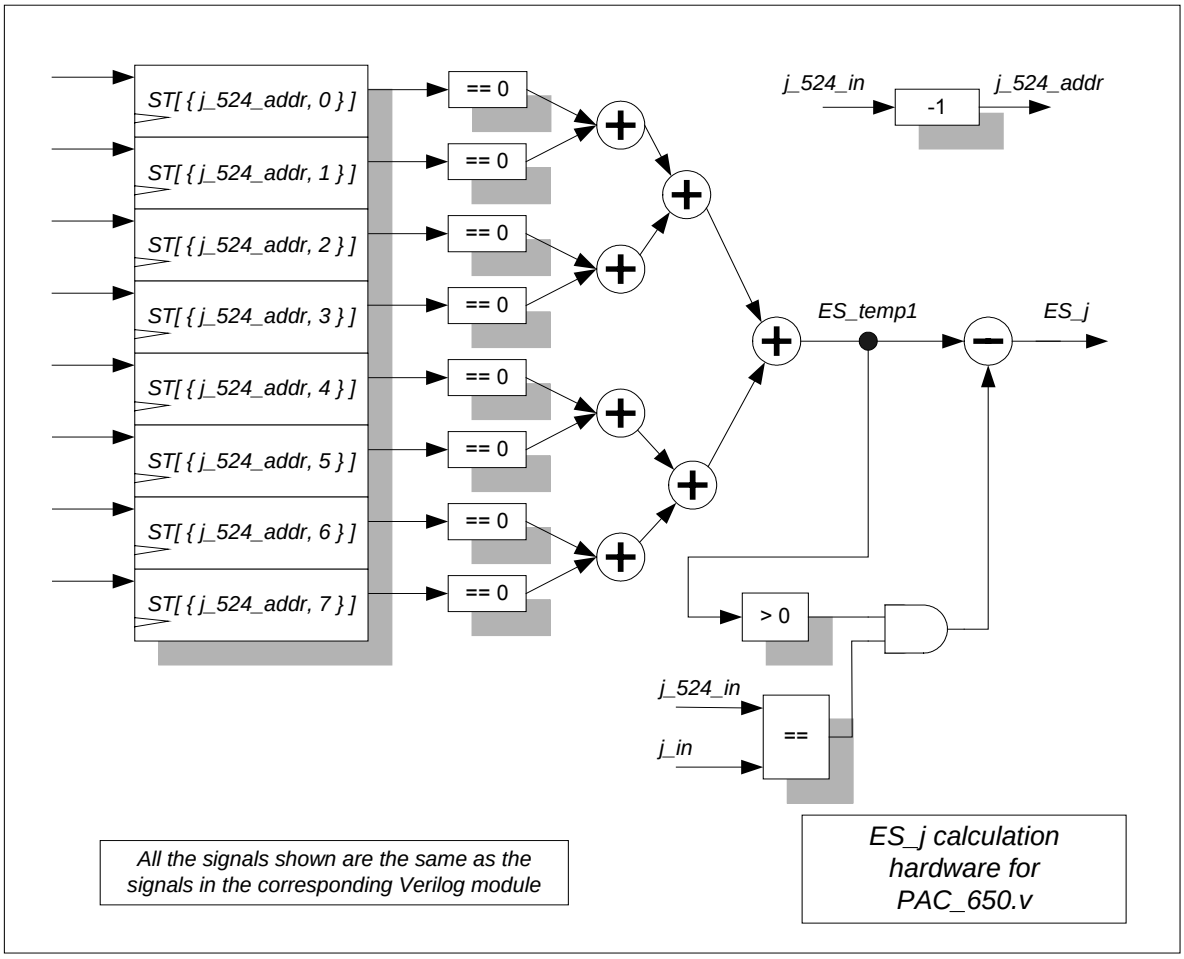



Figure 3.11 *ES_j* calculation in hardware

### 3.2.7 PAC Top Level Design

All of the hardware modules are assembled at the top level to create the PAC. This top-level assembly only consists of modules and wires and is straightforward to derive from the individual modules and their signals. The overall block diagram for this hardware is shown in Figure 3.12. This module assembly occurs in the file *PAC_Full.v*.

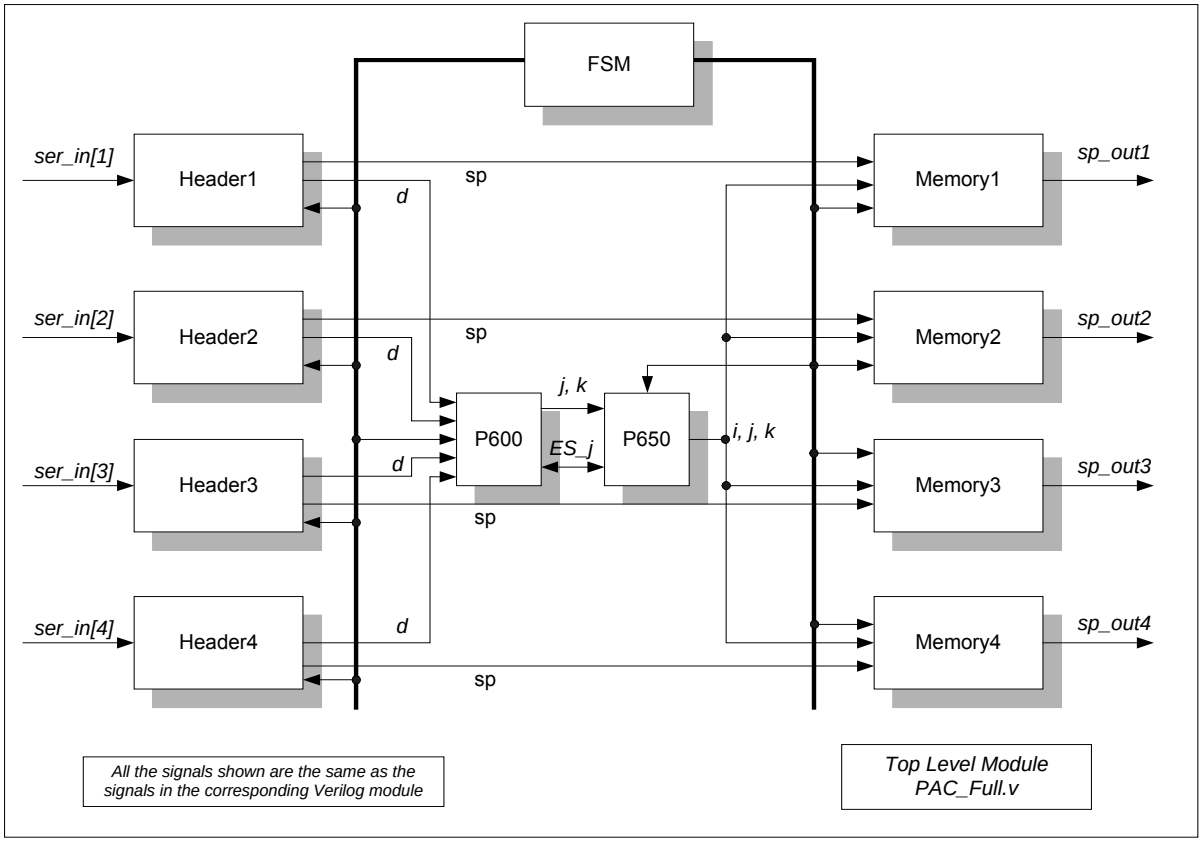



Figure 3.12 Top Level Assembly of the PAC

## 3.3 Input Interconnection Network Design

The Input Interconnection Network (IIN) is the second stage of the pipeline. The main task for this stage of the SW algorithm is to route the packets from each of the four *PAC_Memory.v* modules to one of the eight Memory Modules specified by the packet's *i* parameter. This routing is complicated by the fact that on rare occasions there may be a duplicate *i* value in one switch cycle. The handling of this algorithm to hardware translation problem will be discussed later. For now the IIN can be simply thought of as a crossbar with a one switch-cycle delay. The delay is introduced to stay true to the SW algorithm, which treats the IIN as a pipeline stage.

The packet datapath of the PAC is 64 bits. This datapath size, while not a problem in PAC, poses a problem for the crossbar. The reason is that the crossbar must switch a 64 bit wide datapath. A datapath this wide creates unnecessarily large logic in the crossbar. This size is arbitrary and could be reduced to a 1 bit wide datapath. However, it is best to choose a size that can be written to the Memory Module RAM easily. The size that I decided on was 8 bits. To accomplish this datapath conversion, a module was created that does this and extracts the *i* value for routing.

### 3.3.1 IIN Header Module

The IIN Header Module, *IIN_H.v*, converts the datapath from 64 to 8 bits and extracts the *i* parameter value. It also introduces the pipeline delay the same way that the PAC does: by using 2048 bits of on chip RAM. The same Quartus II generated RAM macro is used to accomplish this as in PAC.

This module uses the control signals generated by a Finite State Machine. This FSM generates all the controls and addresses for each module in the IIN. As the first 64 bits of a packet arrives, the IIN Header Module stores it in the on chip RAM. This word is then read out eight cycles later. The whole word at the output of the RAM is addressed using a mux to select 8 bits at a time. The 8 bit group assigned to *ps_out*. Simultaneously the *i* value is extracted for routing the packet to the appropriate Memory Module. The hardware for this module is shown in Figure 3.13.

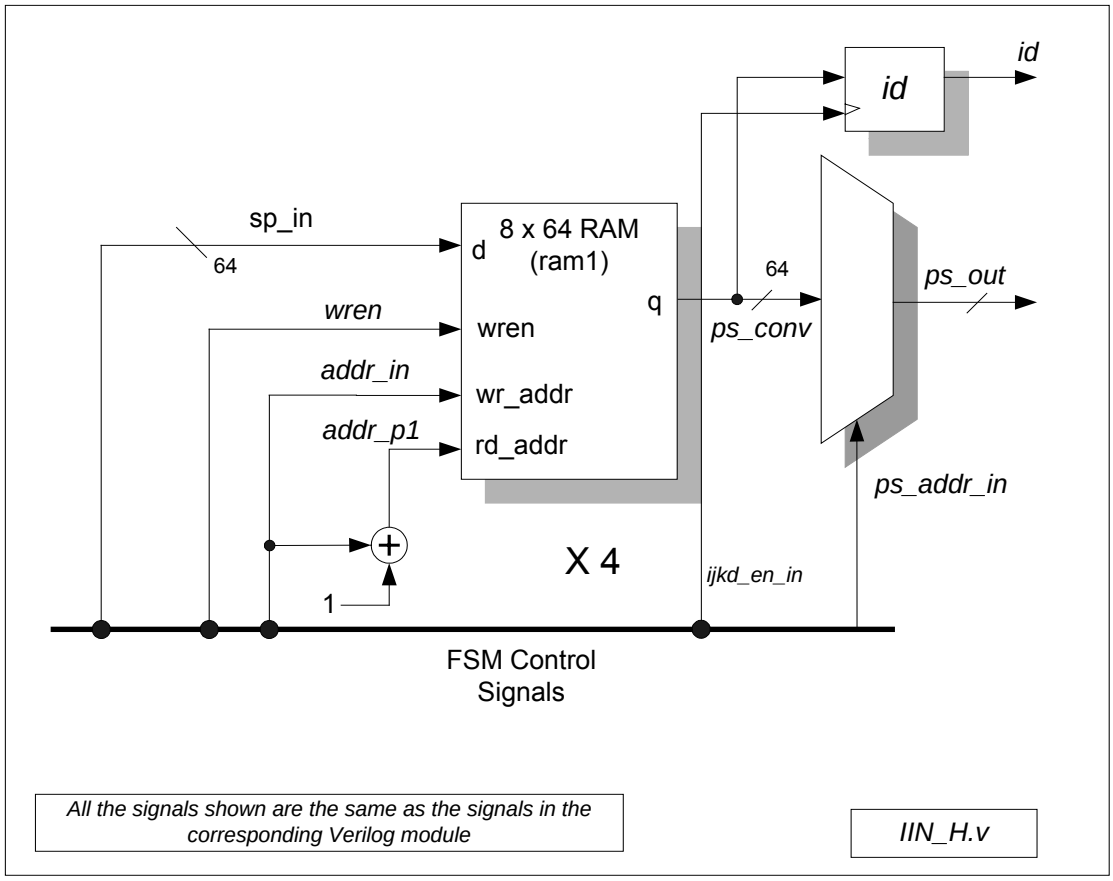



Figure 3.13 IIN Header Module Diagram

### 3.3.2 IIN Crossbar

The main task of the Input Interconnection Network is to route a packet to the appropriate Memory Module. When there are no destination conflicts this can be accomplished with a crossbar. There are multiple ways to implement a crossbar as suggested in [5]. Three of the ways that may be implemented in this FPGA are:

1. A mux for each output
2. Tristate buffer array
3. RAM

The RAM crossbar method unfortunately will not work in parallel like the other two methods. While using Tristate buffers is possible, the simplest way is to just use muxes.

As stated earlier, a crossbar works best when there are no destination conflicts. But, this is not the case with this design. The destination conflict is fortunately predictable in that the design only needs to handle a destination conflict of two. The algorithm also requires that when there is a destination conflict, the two packets need to be written simultaneously to the same Memory Module in the same switch cycle. This is accomplished by doing the normal writes (no destination conflicts) to the memory modules at half the bus speed. When there is a conflict the crossbar toggles between the two packets while the Memory Module writes these two packets to their appropriate location at the normal bus speed.

This design uses priority encoders (PE's) for this task instead of muxes to implement a crossbar. PE's are used in this design because they perform a sequential search, and therefore chooses the first input with a matching *i* value. There are two priority encoders at each output. Each priority encoder performs its "search" in opposite directions. In the case of duplicate *i* values, there will be different packet data on each PE. A mux selects between these two PE's to do the final stage of routing. Most of the time, this mux just selects the first (main) PE. However, when there is a duplicate *i* value, the mux toggles between the main PE and the secondary PE to write both packets simultaneously. Note that the mux could perform this toggle all the time and work correctly (because both PE's will have identical output), but I chose only to toggle in the case of duplicate *i* values. This toggle signal is actually generated by the Memory Module in response to the duplicate *i* signal, *i_dupl_out* from the IIN Crossbar Module. The diagram that corresponds to this module is shown in Figure 3.14.

There is one IIN Crossbar Module, *IIN_CB.v,* connected to each Memory Module. So in this 4x4 switch there will be 8 *IIN_CB.v*'s.

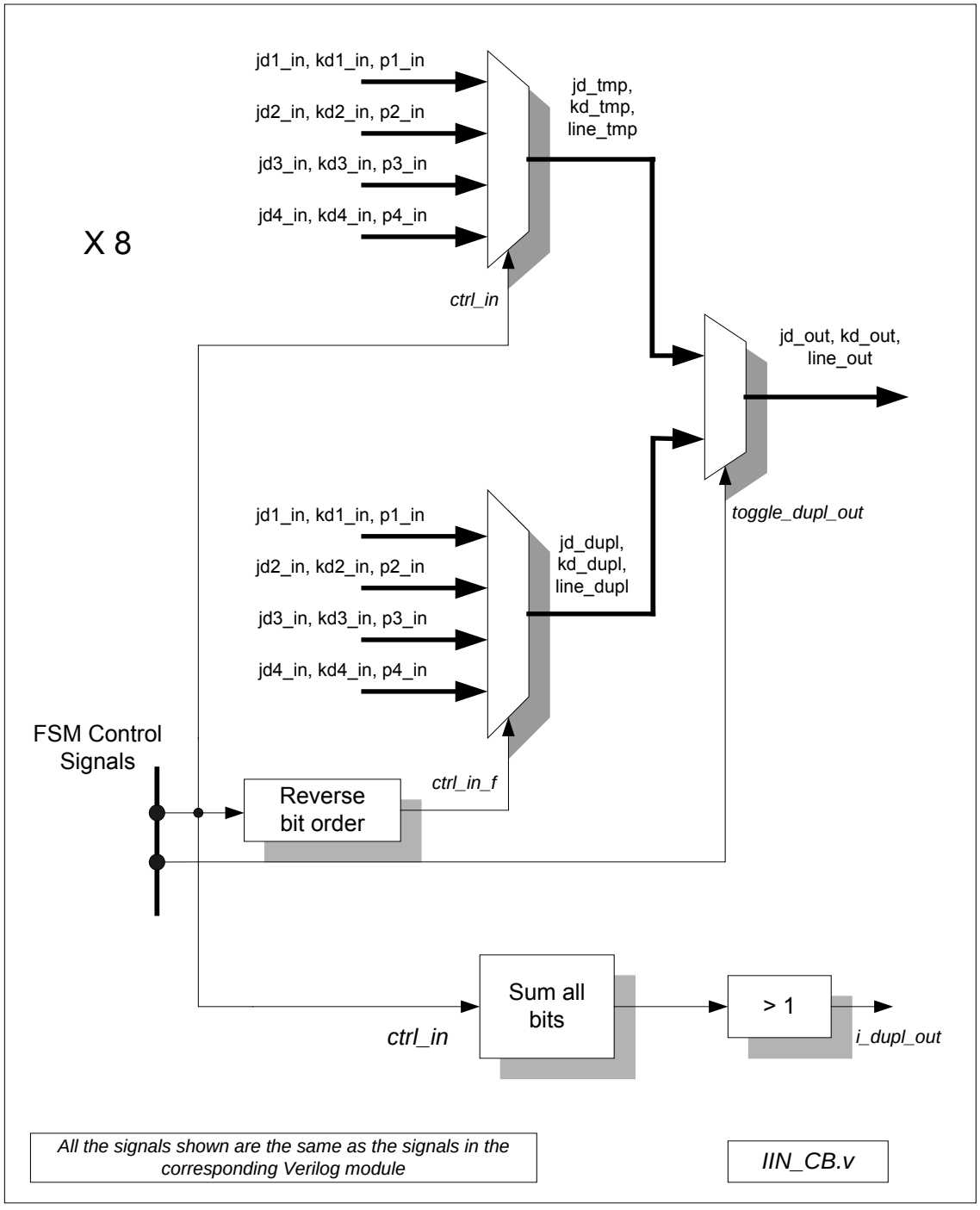



Figure 3.14 IIN Crossbar Design

### 3.3.3 IIN Finite State Machine

Part of the IIN contains a Finite State Machine (FSM) that is modeled after the one in the PAC. It contains a counter and output logic based on the counter to generate addresses and control signals.

There is also extra combinational logic that generates control signals for the Crossbar Modules based on the *i* parameters. Eight control signals, one for each Memory Module, are generated based on the four *i* parameters from the Header modules. These control signals are select lines that controls the actual routing done by the PE's in the Crossbar Module. The hardware block diagram for this module is shown in Figure 3.15.

The control signals are generated by first sending each of the four *i* values from each Header Module to a mux. This mux selects a one-hot 8-bit binary number corresponding to the *i* value. This 8-bit number is divided bitwise so that one bit from each port goes to each of the eight Crossbar Modules. Since there are four inputs, there are four bits total going to each Crossbar Module. This is another one-hot (except when *i* is duplicated) signal for the two PE's in each Crossbar Module. As an example, consider the following:

$$\text{port1\_i} = 3 \xrightarrow{mux} c1 = 00100000$$
$$\text{port2\_i} = 5 \xrightarrow{mux} c2 = 00001000$$
$$\text{port3\_i} = 5 \xrightarrow{mux} c3 = 00001000$$
$$\text{port4\_i} = 6 \xrightarrow{mux} c4 = 00000100$$
$$\text{ctrl1\_out} = \{c1[7], c2[7], c3[7], c4[7]\} = 0000$$
$$\text{ctrl2\_out} = \{c1[6], c2[6], c3[6], c4[6]\} = 0000$$
$$\text{ctrl3\_out} = \{c1[5], c2[5], c3[5], c4[5]\} = 1000$$
$$\text{ctrl4\_out} = \{c1[4], c2[4], c3[4], c4[4]\} = 0000$$
$$\text{ctrl5\_out} = \{c1[3], c2[3], c3[3], c4[3]\} = 0110$$
$$\text{ctrl6\_out} = \{c1[2], c2[2], c3[2], c4[2]\} = 0001$$
$$\text{ctrl7\_out} = \{c1[1], c2[1], c3[1], c4[1]\} = 0000$$
$$\text{ctrl8\_out} = \{c1[0], c2[0], c3[0], c4[0]\} = 0000$$

Notice the control signal *ctrl5_out* has two bits hi (which means there are duplicate *i's)*. This is OK because it is the select line for a priority encoder, which simply searches the select line for the first hi bit. This signal is also flipped bitwise and sent to the second priority encoder in the Crossbar Module. In this case, the main and secondary PE will select different ports. The Crossbar Module detects this and reacts accordingly so that both packets are sent to the Memory Module simultaneously, in an interleaved fashion.

This FSM module contains more than just an FSM. This logic, although not state dependent, had to go somewhere central to the IIN. The FSM is the only single module in the IIN and was therefore the most logical choice.

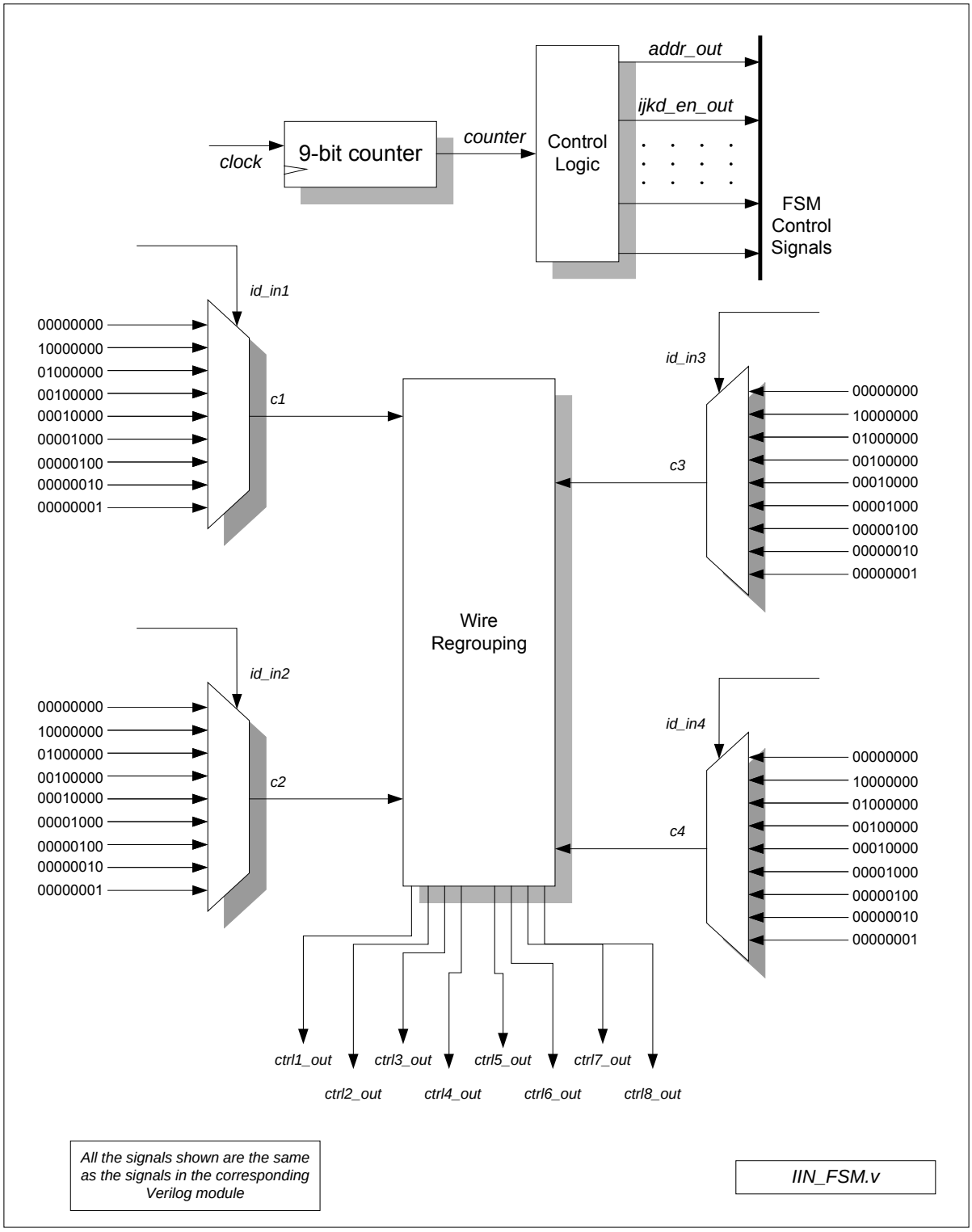



Figure 3.15 IIN FSM Design

### 3.3.4 IIN Top Level Implementation

The upper level IIN design contains one FSM Module, eight Crossbar Modules, and four Header Modules. All of these hardware modules are assembled at the top level to create the IIN. This module assembly occurs in the file *IIN.v*. The overall block diagram for this hardware is shown in Figure 3.16. Some signals, such as *jd_out1..8*, are not shown to prevent clutter.

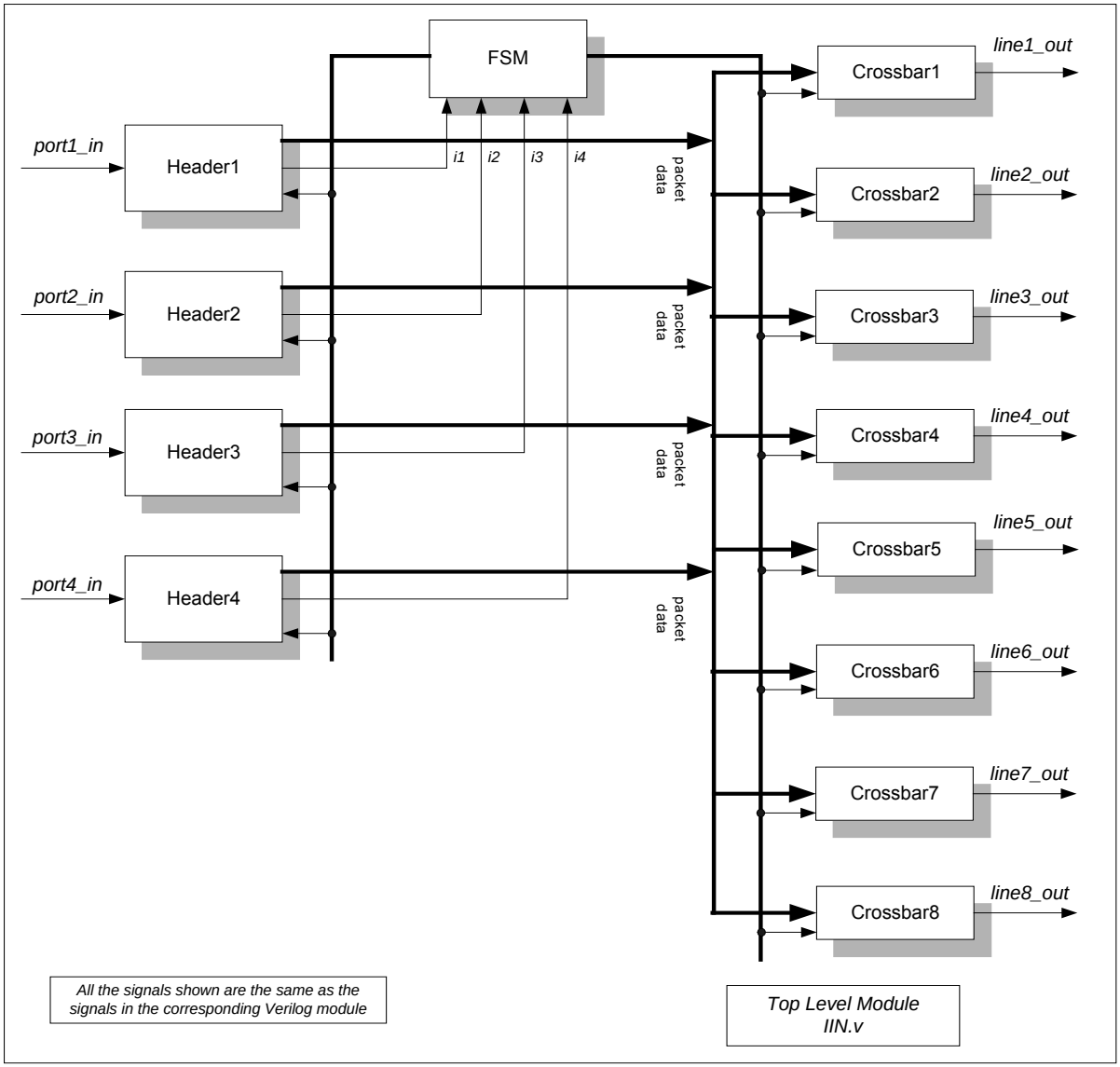



Figure 3.16 IIN Top Level Design

## 3.4 Memory Module Design

So far, the first two stages have been single pipeline stages. This section of the switch actually makes up two pipeline stages. They are the write and read operations, which make up the third and fourth stages of the pipeline respectively. Except for handling the double packet write for duplicate *i* parameters, the design of this section is straightforward.

### 3.4.1 Memory Module Write Operation

The write operation takes place in the Memory Module RAM file *MM_RAM.v*. Recall that the data is arriving in 8-bit words from the IIN Crossbar Module. Since a packet is 64 bytes and sigma = 12, there are 64 x 12 = 768 bytes per Memory Module RAM. There are eight Memory Module RAMs so there are 6144 bytes of RAM in the Memory Module top-level design. The *j* and *k* parameters for an arriving packet have already been extracted in the previous stage (IIN) and are present at each Memory Module RAM input along with the data. This is crucial so that the data may be written immediately. The write depends heavily on the *j* parameter because it points to the slot in the Memory Module that the packet is assigned.

If the *i_dupl_in* signal is raised, then the Memory Module RAM responds by toggling the *toggle_dupl_out* signal. This tells the crossbar to toggle between the main and secondary packet if there is a duplicate *i* parameter for the packets. Toggling between the packet and parameters interleaves the two packets together in one write. Please note that the normal write cycle has been doubled to accommodate this rare occurrence of duplicate *i* parameters.

A write is triggered when a packet arrives. The write address is calculated from an address generated internally by an accumulation register, *w_addr_reg*, as the lower bits concatenated with the *j_addr*. The lower bit of the *w_addr_reg* is the *toggle_dupl_out* signal. The upper eight bits are used in the write address generation. The Output Scan Array is updated in the first (and second if there is a duplicate *i* packet) write cycle from the *j* and *k* values at the inputs.

### 3.4.2 Memory Module Read Operation

The read operation is triggered by the Memory Module FSM. The read address is then internally generated by an accumulation register, *r_addr_reg*, concatenated with *SW_j_addr*. Packet data is not sent out unless the *k* parameter in the *SW_j* slot of the OSA matches the current *SW_k* value. The diagram for this hardware is shown in Figure 3.17.

Consider the case of reading an address that is currently being written. This could cause a conflict and incorrect data. To avoid this conflict, time may be inserted between the writes so that the data can be read. Another way to avoid this conflict is to simply stagger the read and write addresses so that the reads are one address behind the writes. Inserting time between writes seemed to be the simplest way, since there is plenty of time in this design to spare. Sufficient time also already existed between the writes. So eight bytes are written, and then eight bytes are read. This is repeated eight times until the entire 64-byte packet is written and read.

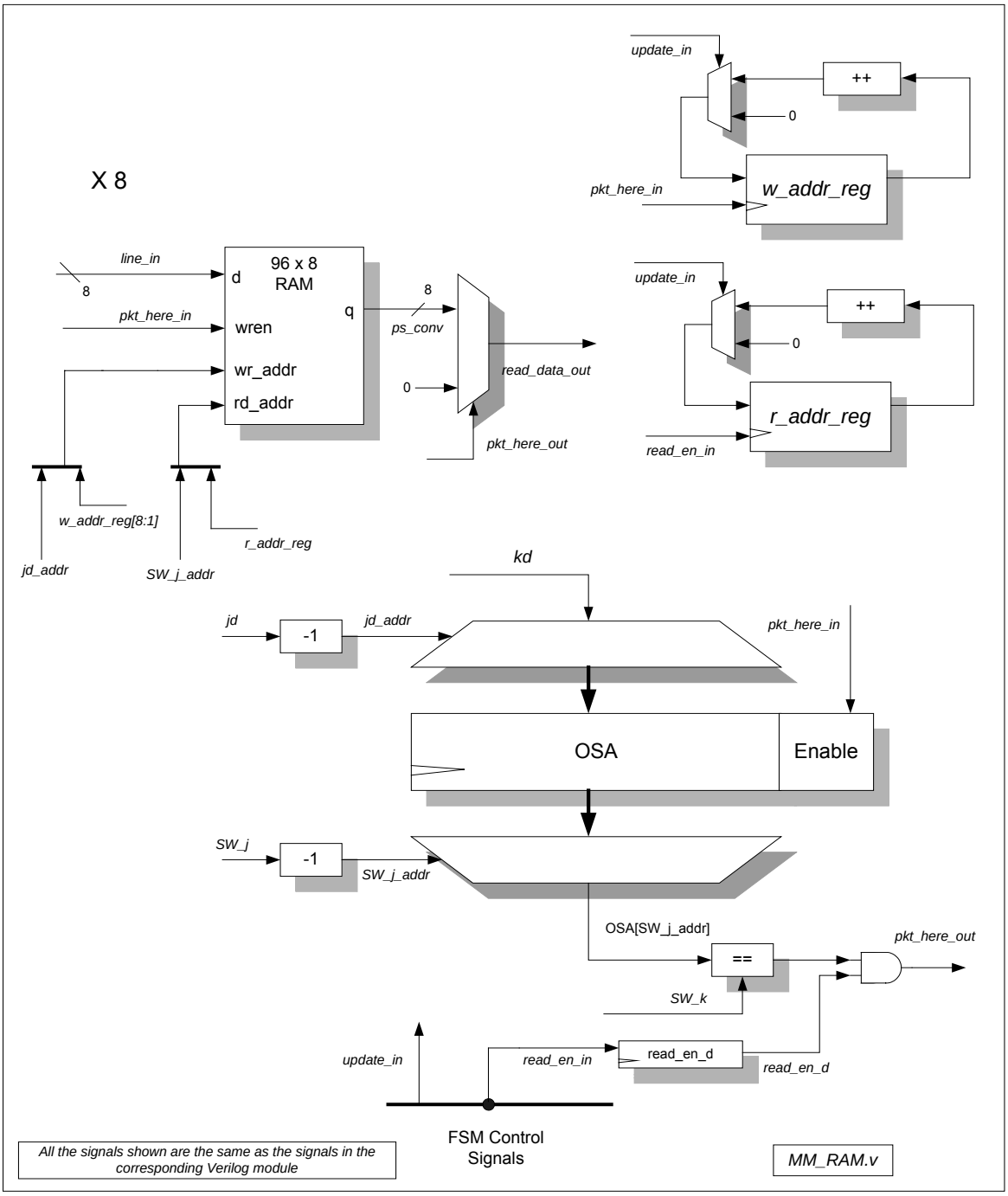



Figure 3.17 Memory Module Design

### 3.4.3 Memory Module FSM

The Memory Module FSM is, like the FSM in the PAC, a counter with output logic. This FSM has a delay register that applies a three switch-cycle delay to the local SW-Counter. This three switch-cycle delay is a bit tricky to derive from the HDL file *MM_FSM.v* or the block diagram (Figure 3.18) because it looks like a two cycle delay. The delay register goes high on the 1st clock cycle of the third switch-cycle. However, the *update* signal does not go high until the beginning of the fourth cycle. The reason there is a delay here is because *delay* will not make its transition until *counter[8:0] = 1*, but *update* needs counter[8:0] to be 0:

*update = (delay && (counter[8:0] == 9'h000));*

The *delay* register is used as a flag to signal that two switch cycles have taken place. The *update* signal, which is based on the *delay* register, is used to update the SW-Counter among other tasks. Since there are three pipeline stages before the Read Operation, the SW-Counter needs to be three cycles behind in the Memory Module. The Memory Module Read operation needs the SW-Counter, but the Write Operation does not.

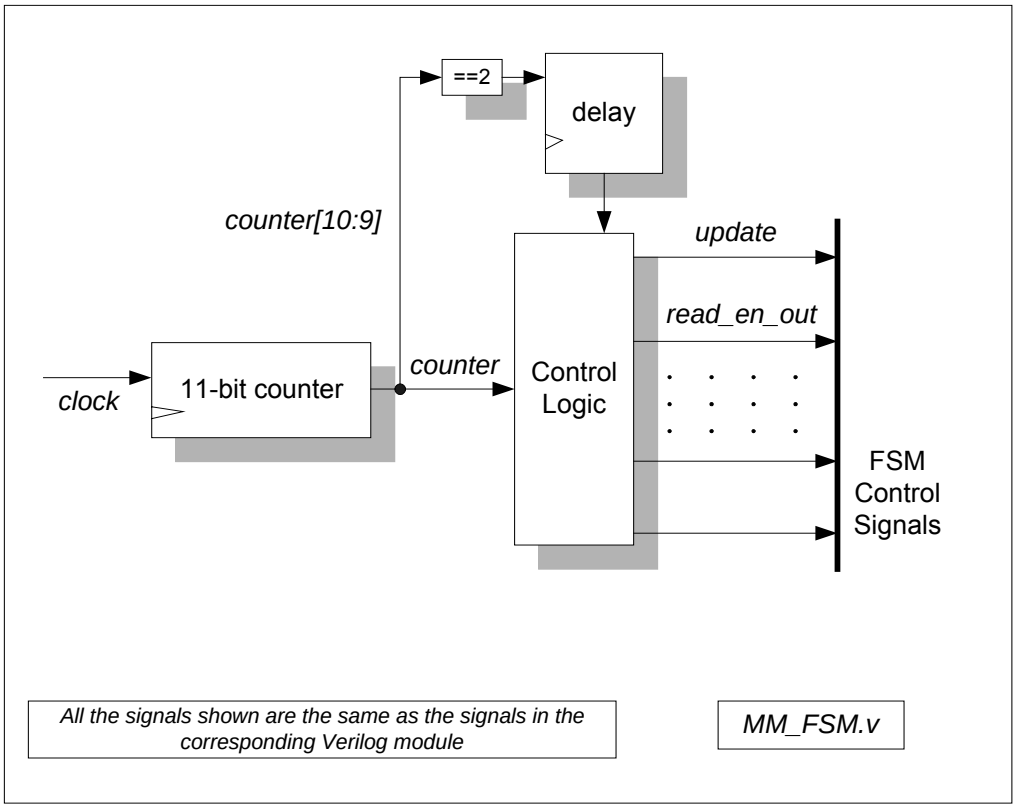



Figure 3.18 MM FSM Design

### 3.4.4 Memory Module Top-Level Design

The Top-Level implementation of the Memory Section consists of an FSM, eight Memory Module RAMs, and one SW-counter as shown in Figure 3.19.

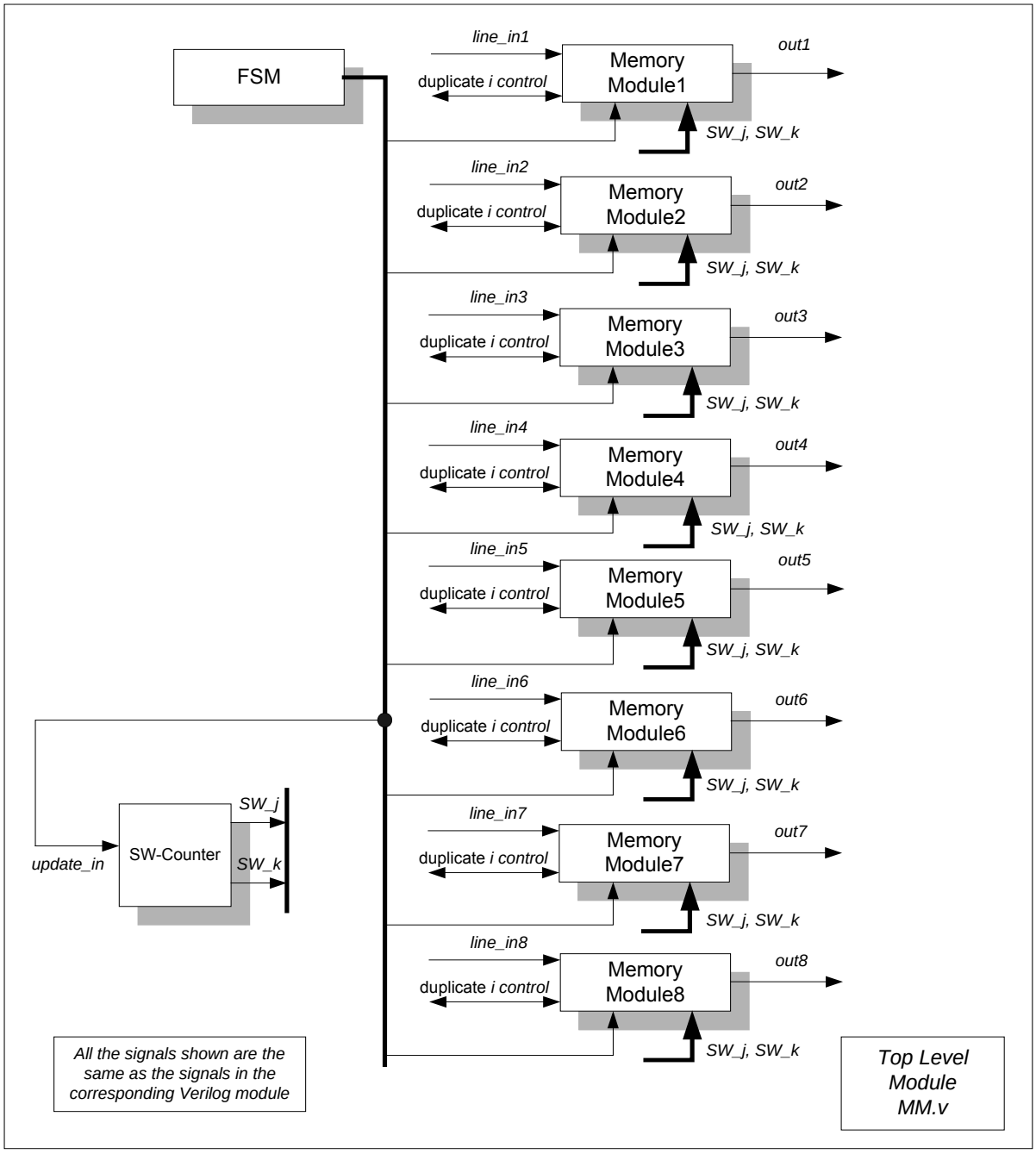



Figure 3.19 Top Level Memory Module Design

## 3.5 Output Interconnection Network

This stage of the pipeline actually does not introduce any type of delay, because it is unnecessary. The OIN is a crossbar that routes the packets to their final destination given by

their respective *d* parameter. There can be no destination conflicts in this stage because the algorithm takes care of this problem.

When packet data arrives, the first byte contains the *d* parameter. This parameter is stored for the duration of the packet data and is simultaneously used to route the packet to its destination port.

The routing for this stage of the switch is done by a crossbar similar to the one in the IIN. The *d* parameter selects a one-hot signal from a mux. This signal is regrouped in a similar way to how the IIN regroups its one-hot signal. The regrouped signal is sent to a priority encoder at the destination port. The destination port is then connected to the appropriate Memory Module.

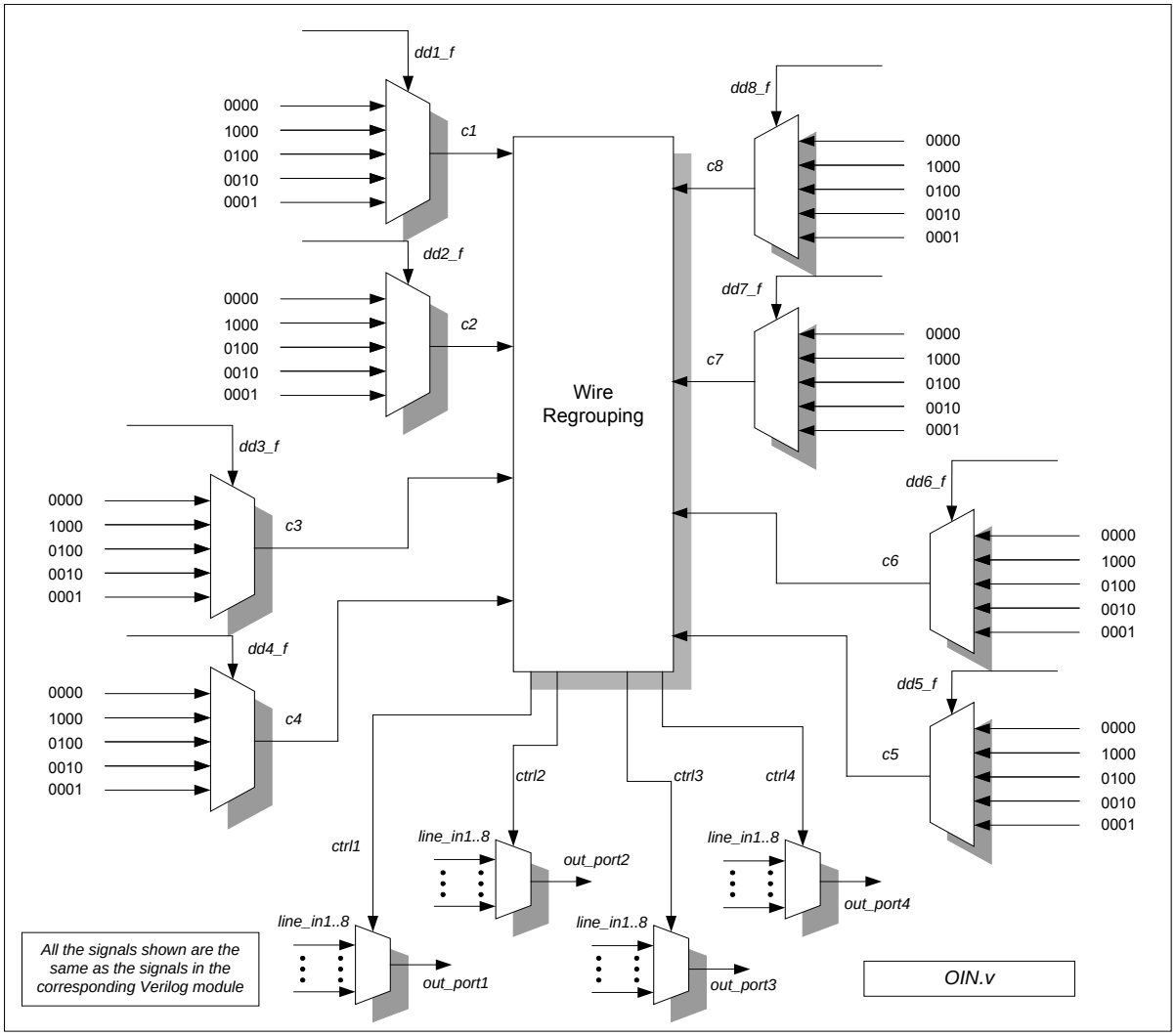



Figure 3.20 OIN Design

## 3.6 Parallel-Serial Conversion

This part of the design is required only for actually viewing the output data in the FPGA design. It is attached to the output of the OIN. As stated earlier in this chapter, there are only 24 channels on the Data Acquisition Card. One way to obtain the packet data is to serialize it so that the number of output ports is minimized to four one-bit ports. There are other ways of doing this such as time-sharing a bus between output ports, but serialization is the most logical choice.

This converter acts as a 64-bit buffer to store the data as it arrives. Serialization is accomplished by writing the arriving packet data to a bank of bit addressable RAM (see Figure 3.21). This RAM is not precompiled on-chip RAM, but just a bank of registers. It is written a byte at the time, but is read a bit at a time. Timing is key for this operation so a 9-bit counter is used. This stage is not part of the SW algorithm.

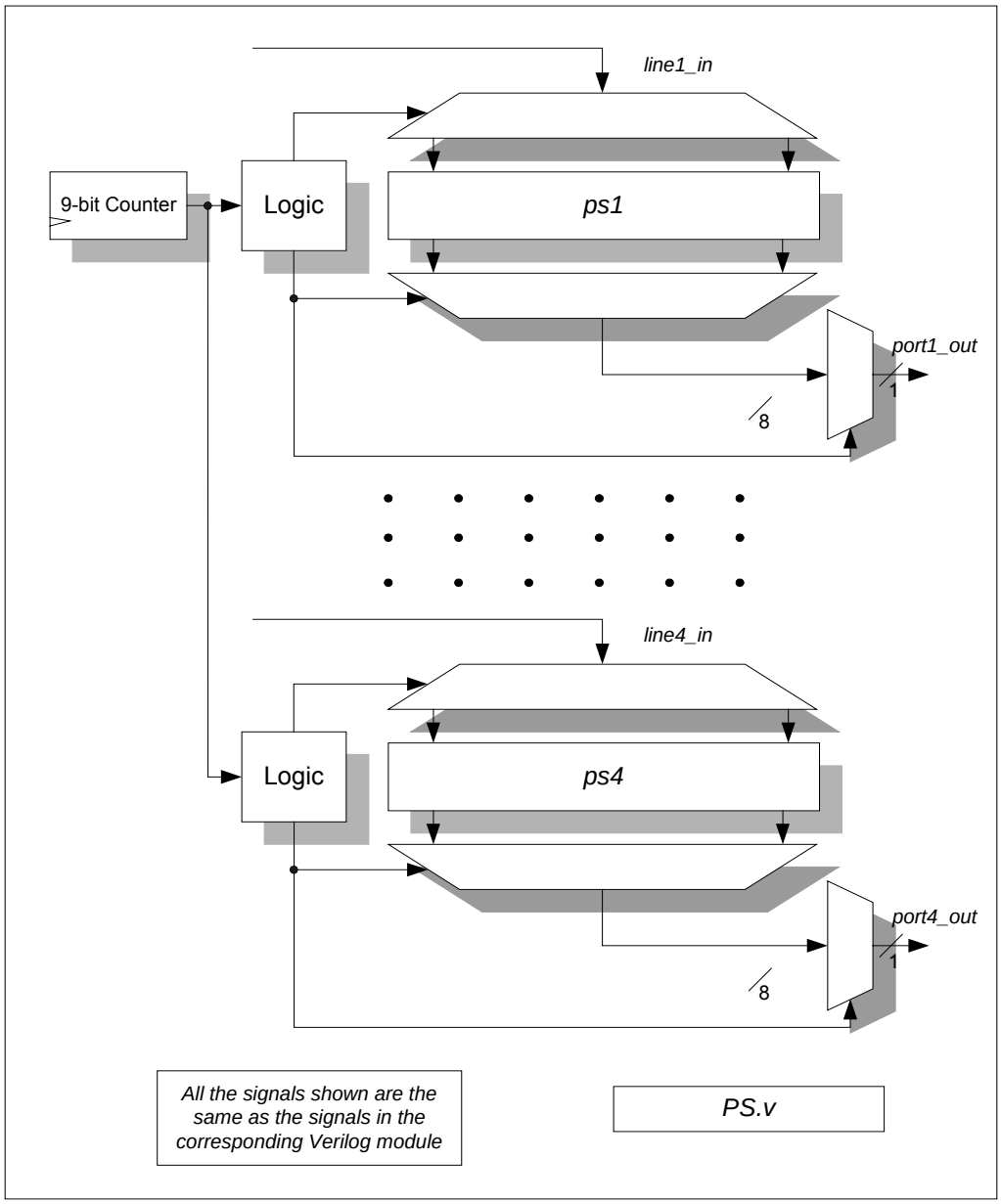



Figure 3.21 Parallel-Serial Conversion

## 3.7 Synthesis

The top-level design for the entire switch is contained in the *SW_Top_Level.v* file. It connects the four sections of the switch together, PAC, IIN, MM, and OIN, to create a 4x4 switch similar to the one shown in Figure 3.1. It also connects the Parallel Serializing Module to the output of the OIN, so that the output is easy to obtain with the available equipment.

All of the input and output signals are connected to their appropriate outputs as seen in the *SW.pin* file generated by the Quartus II synthesizer. This configuration is handled before compilation so that the assigned pins match up to convenient locations on the development board. The clock must be driven externally by the Data Acquisition Card (DAQ) so that the card may easily synchronize with the large quantity of arriving data. The clock cannot be run at full speed since the DAQ must store the data as it arrives.

Since there are no timing requirements, the compiler is run in the default mode. Some of the results of the compilation such as the timing information and the floorplan are displayed in Figure 3.22 and Figure 3.23. The rest of the synthesis is displayed in the Appendix.

```
Warning: Warning: Found pins functioning as undefined clocks and/or memory enables
    Info: Info: Assuming node clock is an undefined clock
Info: Info: Clock clock has Internal fmax of 22.14 MHz between source register
  PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_synch_counter:wysi_counter|sload_path[6] and destination register
  PAC_Full:PAC|PAC_600:P_600|Qd[4][2] (period= 45.165 ns)
    Info: Info: + Longest register to register delay is 44.596 ns
        Info: Info: 1: + IC(0.000 ns) + CELL(0.219 ns) = 0.219 ns; Loc. = LC7_11_Z3; REG Node =
          'PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_synch_counter:wysi_counter|sload_path[6]'
        Info: Info: 2: + IC(1.136 ns) + CELL(1.207 ns) = 2.562 ns; Loc. = LC8_10_Z3; COMB Node = 'PAC_Full:PAC|PAC_NCFSM:NCFSM|reduce_nor_30~117'
        Info: Info: 3: + IC(0.300 ns) + CELL(1.207 ns) = 4.069 ns; Loc. = LC2_10_Z3; COMB Node = 'PAC_Full:PAC|PAC_NCFSM:NCFSM|i~2'
        Info: Info: 4: + IC(0.324 ns) + CELL(1.192 ns) = 5.585 ns; Loc. = LC4_9_Z3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|Select_112_rtl_61~0'
        Info: Info: 5: + IC(0.294 ns) + CELL(0.486 ns) = 6.365 ns; Loc. = LC2_9_Z3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|Select_112_rtl_61~1'
        Info: Info: 6: + IC(1.315 ns) + CELL(1.082 ns) = 8.762 ns; Loc. = LC2_1_Z3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|Mux_5_rtl_2172~0'
        Info: Info: 7: + IC(0.287 ns) + CELL(0.486 ns) = 9.535 ns; Loc. = LC1_1_Z3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|Mux_5_rtl_2172~1'
        Info: Info: 8: + IC(0.287 ns) + CELL(0.486 ns) = 10.308 ns; Loc. = LC10_1_Z3; COMB Node = 'rtl~2160'
        Info: Info: 9: + IC(2.367 ns) + CELL(1.521 ns) = 14.196 ns; Loc. = LC6_2_W3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|add_11~4COUT'
        Info: Info: 10: + IC(0.000 ns) + CELL(1.000 ns) = 15.196 ns; Loc. = LC7_2_W3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|add_11~5'
        Info: Info: 11: + IC(0.273 ns) + CELL(1.082 ns) = 16.551 ns; Loc. = LC9_2_W3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|reduce_nor_62~12'
        Info: Info: 12: + IC(0.283 ns) + CELL(1.082 ns) = 17.916 ns; Loc. = LC6_3_W3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|j_f[1]~25'
        Info: Info: 13: + IC(0.287 ns) + CELL(1.192 ns) = 19.395 ns; Loc. = LC4_3_W3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|i~567'
        Info: Info: 14: + IC(0.297 ns) + CELL(1.082 ns) = 20.774 ns; Loc. = LC9_3_W3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|j_524_addr[1]~1'
        Info: Info: 15: + IC(2.636 ns) + CELL(1.082 ns) = 24.492 ns; Loc. = LC10_2_R3; COMB Node =
          'PAC_Full:PAC|PAC_650:P_650|Mux_137_rtl_407_rtl_503_rtl_599_rtl_1816~0'
        Info: Info: 16: + IC(0.273 ns) + CELL(0.486 ns) = 25.251 ns; Loc. = LC9_1_R3; COMB Node =
          'PAC_Full:PAC|PAC_650:P_650|Mux_137_rtl_407_rtl_503_rtl_599_rtl_1816~1'
        Info: Info: 17: + IC(0.259 ns) + CELL(1.082 ns) = 26.592 ns; Loc. = LC5_1_R3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|reduce_nor_139~28'
        Info: Info: 18: + IC(2.626 ns) + CELL(1.207 ns) = 30.425 ns; Loc. = LC7_11_U3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|reduce_nor_139~130'
        Info: Info: 19: + IC(1.162 ns) + CELL(1.082 ns) = 32.669 ns; Loc. = LC4_10_U3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|add_144~0'
        Info: Info: 20: + IC(1.210 ns) + CELL(1.082 ns) = 34.961 ns; Loc. = LC1_9_U3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|add_145~57'
        Info: Info: 21: + IC(0.273 ns) + CELL(1.207 ns) = 36.441 ns; Loc. = LC5_9_U3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|i~52324'
        Info: Info: 22: + IC(0.304 ns) + CELL(1.192 ns) = 37.937 ns; Loc. = LC9_8_U3; COMB Node = 'PAC_Full:PAC|PAC_650:P_650|add_55~15'
        Info: Info: 23: + IC(0.297 ns) + CELL(1.082 ns) = 39.316 ns; Loc. = LC3_8_U3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|LessThan_101_rtl_607~19'
        Info: Info: 24: + IC(0.269 ns) + CELL(1.207 ns) = 40.792 ns; Loc. = LC8_8_U3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|i~1'
        Info: Info: 25: + IC(2.748 ns) + CELL(0.486 ns) = 44.026 ns; Loc. = LC7_14_Z3; COMB Node = 'PAC_Full:PAC|PAC_600:P_600|i~1217'
        Info: Info: 26: + IC(0.310 ns) + CELL(0.260 ns) = 44.596 ns; Loc. = LC6_14_Z3; REG Node = 'PAC_Full:PAC|PAC_600:P_600|Qd[4][2]'
        Info: Info: Total cell delay = 24.779 ns
        Info: Info: Total interconnect delay = 19.817 ns
```

Figure 3.22 Synthesis Timing Output

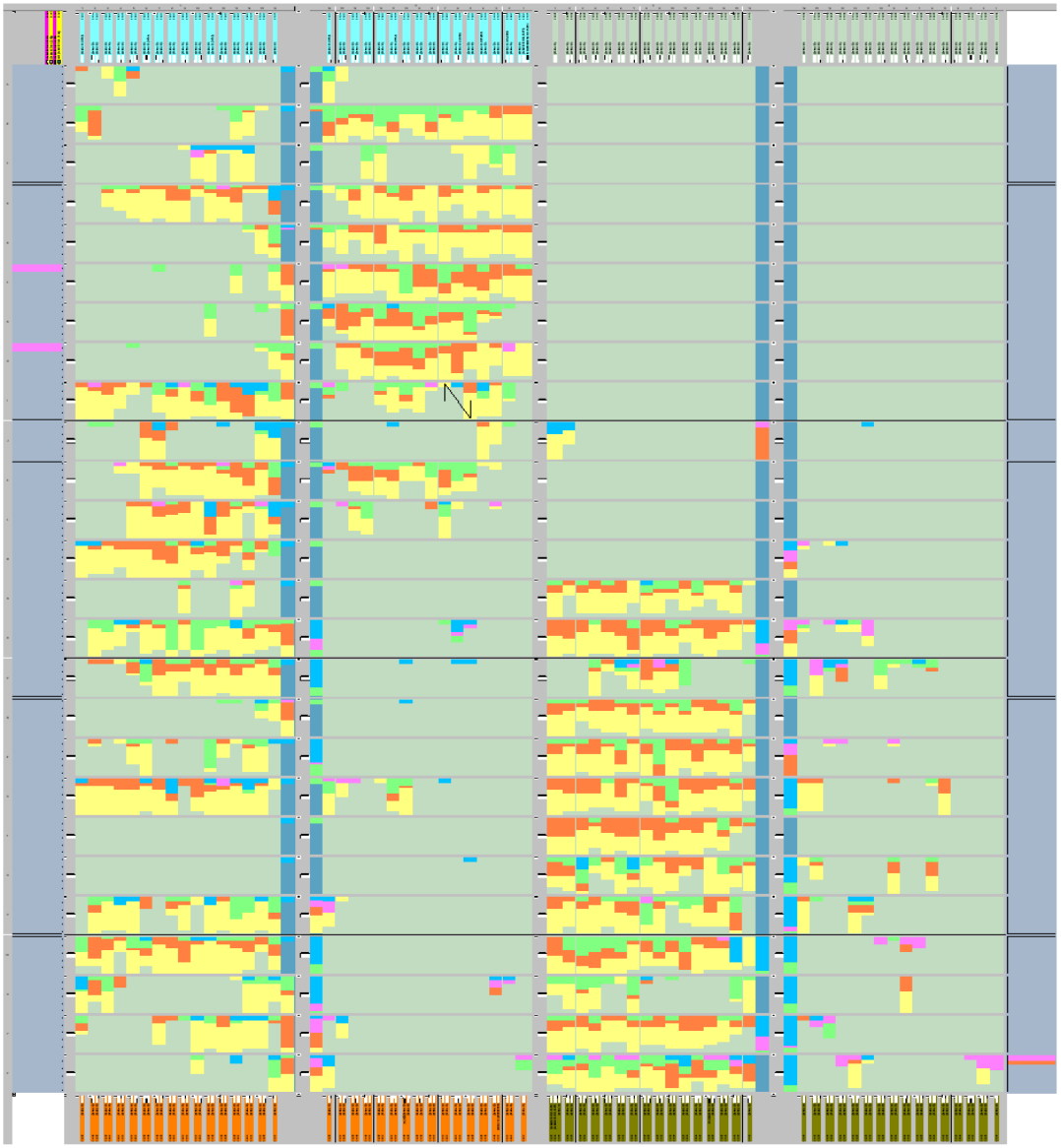


Figure 3.23 Floorplan of the Synthesized Circuit

# Chapter 4

# Hardware Scalability

This chapter will discuss the Sliding-Window hardware scalability constraints section-by-section. The implementation covered in this project is a four port SW-switch. As port quantity increases in some shared memory architectures, memory bandwidth and centralized control can pose severe limits. One of the goals of this project is to evaluate the limits of hardware as port quantity is increased.

The assumptions for the input traffic made in Chapter 3, hold true for this chapter, mainly that the traffic consists of 64 byte packets.

## 4.1 Parameter Assignment Circuit Constraints

The Parameter Assignment Circuit (PAC) modules can be divided into two different types of hardware: parallel hardware and sequential hardware. Parallel hardware is duplicated as port quantity increases. The Header Modules and PAC-Memory are essentially parallel modules. These modules are not likely timing bottlenecks for the PAC because they share the port load. As port quantity increases, so does the number of PAC-Memory and Header Modules. Therefore, other than added area, these modules are not a scalability concern.

Sequential hardware takes on more duties as port quantity increases. In the PAC, the Finite State Machine (FSM), Processor 1 (P600), and Processor 2 (P650) can be considered sequential modules. Due to their sequential nature, these modules are where bottlenecks are

most likely to occur. The parallel modules must wait on the sequential modules to complete their tasks before they can continue. An exception is the FSM. Although it is the central control for the PAC, as port size increases it only needs to send out more signals. Since this module is a counter and output logic, it is not a likely bottleneck that needs considering.

The two PAC Processors, P600 and P650, operate in a pipeline fashion on the group of packets at the input ports during a single switch cycle. This is accomplished in a packet-by-packet manner. This becomes a bottleneck for large port quantities. For example, consider a 512x512 SW-packet switch. The processors would need to complete 512 *j, k* parameter assignments per switch cycle. To find out if this is possible a data rate must be assumed [1]:

1. OC-3 data rate (155 Mbps)
2. Assume 64 byte (512 bit) packets
3. 512 bits / 155 Mbps = 3.3 μs per switch cycle.

This data rate assumption yields a 3.3μs time limit to assign all the parameters for each packet per switch cycle, regardless of port quantity. This timing constraint will be used for the timing bottleneck analyses.

### 4.1.1 Processor 1 Scalability

From Figure 3.8, it is apparent that P600 has a few extended logic chains. One lengthy logic chain starts with the signal *d_ctrl* from the PAC FSM. This selects the *d* value which is used to select *LC_j[d]*, which goes through a mux-incrementer to a comparator. The output of this comparator is the select line for the *k_mo* mux, which flows through two more selector muxes to feed *j_f* (also *j_524*). This signal loops through P650 for the *ES_j* calculation. P650 uses the *j_524* signal to address a *ST* vector. The elements of this vector

that are zero are counted by a set of adders. This value, travels through another 2:1 mux back to P600 for comparison to *r*. This result is used to determine if a packet is dropped, which in turn enables the input to all of the parameter registers such as *jd, kd, d,* etc. This logic chain increases very little for larger port quantities. In summary, the logic chain that determines if a packet is dropped has many dependencies. This critical path fortunately does not increase very much with increased port quantity. This path is consistent with the critical path in the timing output of the synthesis (See Appendix). This is because all the decisions and comparisons that are made in determining packet parameters do not increase in quantity. In future work it may be pipelined to multiple stages.

The place this logic chain does increases in length is in the adder stages for determination of *ES_j*. These stages increase approximately logarithmically. As an example, compare this part of the chain for a 4x4 switch to a 512x512 switch. The 4x4 switch, with N = 4 and m = 2N, needs 8 memory modules, which means the Scan Table vector is 8 elements long. This requires $\log_2(8) = 3$ stages of adders to calculate *ES_j,* which is shown in Figure 3.11. Each stage needs a larger datapath since they are summing all the previous stage outputs. The final adder must have two 3-bit inputs and a 4-bit output or $\log_2(8)+1 = 4$. Consider the 512x512 switch. It needs, with N = 512 and m = 2N, 1024 memory modules. This results in a 1024 element *ST* vector. The output of each zero compare is only one bit for the 1st stage, like the 4x4 switch. However, this port quantity of N = 512 requires $\log_2(1024) = 10$ stages of adders. The last adder stage will require two 10-bit inputs and one $\log_2(1024) + 1 = 11$-bit output.

The 2:1 muxes stay the same in this *LC_j[d]* logic chain. The rest of the logic chains for P600 stay mostly the same for increasing N. However there is a similar adder

issue for finding *r*. The set of adders for *r* is increased at $\log_2$ (N) instead of $\log_2$ (m), so it will have one stage less than the adders for *ES_j.* These adders (for finding *r*) are also not in the critical path like the ones for *ES_j,* so they can effectively be ignored.

The critical path in P600 (which loops through P650) is considerable, especially since this processor is sequential. None of the logic in this processor was minimized for this design. It currently follows the algorithm as closely as possible so that it is simple to debug. This is one reason the path is so long. Since each packet assignment depends on the parameters of the previous packet, pipelining would be tricky but not impractical. This pipelining could be the subject of future work. However, the critical path in this processor is not quite as limiting as the one in P650 could be. This is demonstrated in the next section.

### 4.1.2 Processor 2 Scalability

Processor 2 (P650) does not appear at first to have any lengthy critical paths in Figure 3.10. Consider the output of the vector *ST[j_addr]*. This signal travels through a comparator, logic, and into the select line of priority encoder PE1 to determine the *i* parameter. A priority encoder does not scale quite as well as other hardware.

One way to describe a Priority Encoder in an HDL such as Verilog is with a string of *if - else if* statements as given in the sample code:

```
always@(A)
  if(A[3])Y=3;
    else if (A[2]) Y=2;
      else if (A[1]) Y=1;
        else if(A[0])Y=0;
          else Y=2'bxx;
```

This statement describes a 4-input priority encoder. This best illustrates the synthesized logic for a priority encoder because the worst case traverses 4 if-else statements. This is correlated to the synthesized logic because the worst case must traverse approximately the same number of gates. For a 16-input priority encoder, Synopsys synthesizes an approximately 18 gate deep critical path as illustrated in Figure 4.1. The registers shown are for timing analysis and not present in an actual priority encoder.

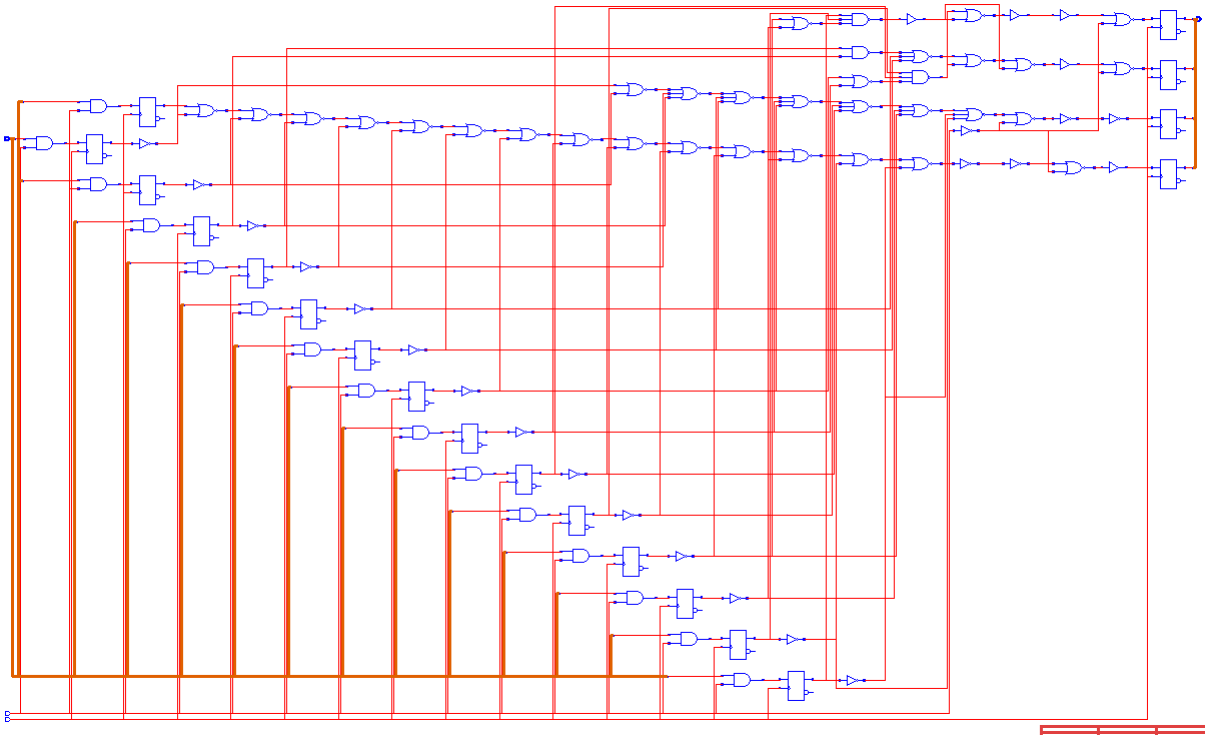


Figure 4.1 Synopsys Plot of a 16 input priority encoder.

For a 512x512 switch, this priority encoder is 1024 select bits wide with 1024 inputs. The above example suggests that if the size were increased to 1024, there would be an approximate 1024 OR-gate critical path. Fortunately, Synopsys parallelizes the hardware for larger priority encoders.

To find out how much this parallelization helps, consider the 1024 input priority encoder case suggested above. For this analysis, Synopsys can be used to synthesize the

logic and obtain timing information. Since an exact clock cycle is not needed, the method used is simply to find the lowest clock cycle that will meet timing constraints under medium effort. This logic chain contains a minimal amount of other logic, so the rest of the circuit is ignored.

To implement the 1024 input Priority Encoder in Verilog, it would be tedious to use the above-mentioned method of *if-else* statements. Another option is to use a *for-loop* of *if* statements. According to [6], this behavioral model implies the same logic. The priority encoder synthesized for this project had to be broken down to a set of two 512 input priority encoders, with a selector mux as shown in 4.3. This is due to a maximum constraint in Synopsys for *for-loops*.

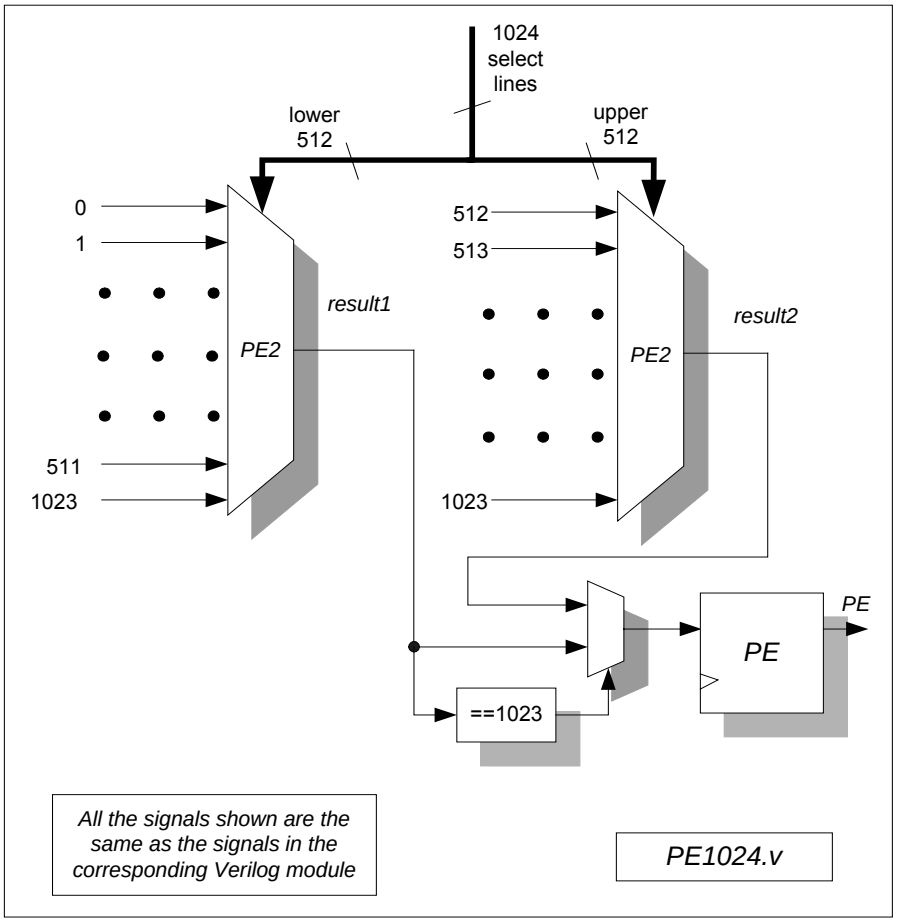



Figure 4.3 1024 Input Priority Encoder Design

An example of the *for-loop* used to create the priority encoders for this compilation is shown below:

```
always @ (value[511:0])
    begin
      result1=10'b1111111111;
      for (k=511; k>=0; k=k-1)
        if(!value[k])
          result1=k;
    end
```

This design was compiled using Synopsys Design Analyzer and the .25µ library. The results of this synthesis show that at medium effort, a 12ns clock is achievable. The other results of this synthesis, the synthesis scripts, and Verilog code are contained in the appendix.

The .25µ library is relatively old technology. Access to other cell libraries such as .09µ is not practical for the scope of this project. This is not a problem because timing estimates may be made for a compiled .09µ design by making a simple assumption: gate length is proportional to delay. This assumption is used in the following equation:

$$\frac{12ns}{.25\mu m} = \frac{\mathrm{T}ns}{.09\mu m}$$
$$\Rightarrow \mathrm{T} = 4.32ns$$
$$\Rightarrow 231MHz$$

The results of the above estimation yield a 4.32ns clock. The timing requirement for a switch at OC-3 (155Mbps) is about 3.3µs as mentioned above. This value is constant regardless of port quantity. However, the port quantity does affect the time that the PAC Processors have to complete each packet's parameter assignments. This is because at OC-3, every 3.3µs a packet arrives at each of the ports (assuming a full load). For the 512x512 switch mentioned above, the time to complete each packet parameter assignment is:

$$\mathrm{Time} = \frac{\frac{1}{\mathrm{Rate(bits/sec)}}\mathrm{packetsize(bits)}}{\mathrm{N}}$$
$$\Rightarrow \frac{\frac{1}{155\mathrm{E6(bits/sec)}}512\mathrm{(bits)}}{512\mathrm{(N)}} = 6.4516\mathrm{ns}$$

This is larger than the estimated clock cycle by over 2ns per packet. These results show that with .09µ technology, the bottleneck of the PAC could easily handle a 512x512 switch at OC-3 with time to spare. The data throughput achieved through this design would be 79.36 Gbps.

P650 and P600 are the only sequentially operating portions of the hardware for the entire switch, which means the *i* assignment is the slowest hardware for large port quantities.

Speeding up this portion of the switch would effectively speed up the whole switch. There is promising work presented in [7] that suggests that large priority encoders (256-bits) could be implemented using CMOS techniques for dramatic gate delay improvements. This work deals with .6um technology. [7] states that using a three-level folding and three-level look ahead (TLF-TLLA) technique, a speedup of 8 compared to conventional priority look-ahead CMOS designs is achievable. In Table 4.1 from [7] it is apparent why this is so: the gate delay has been reduced from 67 to 8, which is the $\log_2$ of 256.

Table 4.1 256-bit Priority Encoder Comparison from [7]

| Design | 3-Level Look Ahead | 3-Level Folding | Logic Chain Depth (Gates) | Normalized Gate Delay |
|---|---|---|---|---|
| Priority Look-Ahead | | | 67 | 1 |
| PE1 | x | | 33 | 0.49 |
| PE2 | x | x | 8 | 0.12 |

Figure 4.4 from [7] shows the experimental 256-bit priority encoder, which yielded a 116 MHz clock. Please note that *PE1* and *PE2* in Table 4.1 are unrelated to the Verilog priority encoders *PE1* and *PE2* presented in this thesis. Table 4.2 compares a 1024-bit (for a 512x512 switch) TLF-TLLA priority encoder to the Synopsys synthesized 1024-bit priority encoder. Note that the logic depth is severely reduced, because according to [7] the gate depth is $\log_2$(bits).

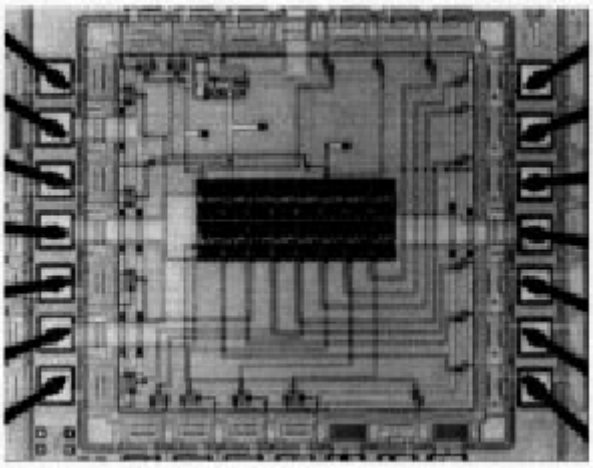


Figure 4.4 The 256-bit TLF-TLLA PE used in the experiment in [7]

If the results of these experiments are scaled to .09μm using the method presented earlier, then a clock cycle using this design may be estimated.

$$\frac{8.62ns}{.6\mu m} = \frac{\mathrm{T}ns}{.09\mu m}$$
$$\Rightarrow \mathrm{T} = 1.293ns$$
$$\Rightarrow 773.4MHz$$

A conservative clock estimate using this design is 400 MHz (2.5 ns), since the priority encoders must travel through an eight input mux, and other minimal logic in P650. Also, the critical paths (that are not as dependent on port quantity) in P600 become a factor and must be considered.

Table 4.2 Comparison of priority Encoders.

| 1024-bit Priority Encoder | Clock Rate | Logic Chain Depth (Gates) (Minimum) |
|---|---|---|
| TLF-TLLA Priority Encoder | 2.5ns | 10 |
| Synthesized Priority Encoder | 12.5ns | 66 |

Consider a 1024x1024 switch. A 2048-bit PE would be needed in P650 to accomplish this. Putting eight of the 256-bit PE's from [7] in parallel could do this. Then the output would be based on the result of an eight input mux that selects the correct value from these eight PE's. This would be similar to the method suggested in Figure 4.3 that uses two PE's. The timing constraints also change for a switch with this port quantity:

$$\text{Time} = \frac{\frac{1}{\text{Rate(bits/sec)}}\text{packetsize(bits)}}{\text{N}}$$

$$\Rightarrow \frac{\frac{1}{155\text{E6(bits/sec)}}512\text{(bits)}}{1024\text{(N)}} = 3.2\text{ns}$$

With a 2.5 ns clock, P650 will be able to handle 1024 *i* assignments per switch cycle with 0.7 ns to spare for each packet assuming an OC-3 data rate. This means that a throughput of 158.72 Gbps may be achievable with this design.

In conclusion, the Priority Encoder used in the *i* parameter search will not be a factor for large port quantities if the TLF-TLLA PE presented in [7] is used. The critical path for this PAC design will then most likely be in the P600-P650 logic loop discussed above. It is possible to pipeline this critical path, and this optimization can take place in future work.

The remainder of the switch hardware such as the IIN, OIN, and the Memory Modules are not sequential but parallel, though this hardware will be analyzed in the remainder of this chapter.

## 4.2 IIN/OIN Scalability

The IIN and OIN implemented in this project use the combinational logic available on the FPGA configured as priority encoders. This design uses eight 4:1 muxes in the IIN and four 8:1 muxes in the OIN. Using this design for a large port quantity, such as the 512x512 switch, would require 1024 Priority Encoders with 512 inputs each in the IIN. This may not pose a timing constraint, but it would definitely be an area and power constraint. This method would be impractical for large port quantities.

Another crossbar design presented in [5] is to use a grid of tri-state buffers such as in Figure 4.5. For a 512x512 switch, this grid would require about 512 x 1024 = 524,288 tri-state buffers to implement the IIN design (per bit of datapath). However, this method is known to have a poor throughput compared to the combinational logic crossbar. For the 512x512 switch operating at OC-3 (155 Mbps), this may not be a problem.

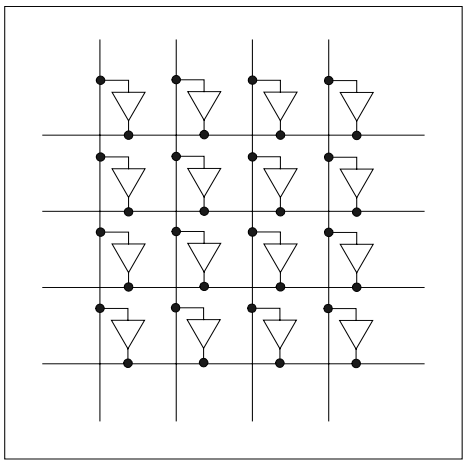


Figure 4.5 4x4 Tri-state Buffer Crossbar Implementation

The most promising solution is to use VLSI crossbar designs. One such design is the one presented in [8]. This design achieves 5.3 Gbps on each path through a 128x128 crossbar.

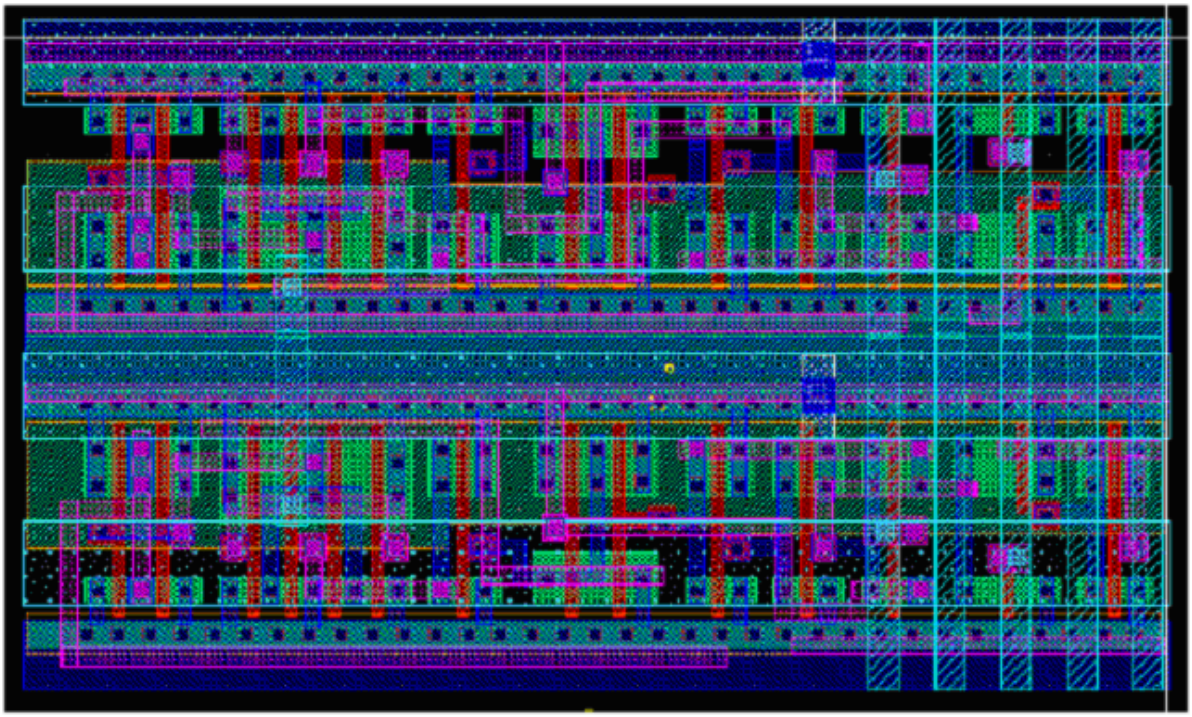


Figure 4.6 Crossbar cell layout from [8]

A solution such as this is more practical for large switches than the combinational logic based mux crossbar mentioned above because it would yield higher data rates and smaller area.

## 4.3 Memory Module Scalability

One of the excellent features of the SW Packet Switch algorithm is that memory bandwidth is not shared between the ports, but distributed among separate Memory Modules. The data bandwidth for one Memory port is the same as the data bandwidth on any given input port for large switch sizes. For this reason, the data bandwidth requirements may be considered independently of port quantity.

The 4x4 switch implemented for this project has a 96-packet memory (49,152 bits) distributed among eight memory modules. Each Memory Module has a read and write port.

For this design, the Memory Modules had to have a data bandwidth of twice the input data rate. The reason for this is that there is a remote chance of a duplicate *i* parameter assignment in a given switch cycle. This forces the Memory Module to write both packets in the same switch cycle or drop one. As discussed in Chapter 3, this design writes both packets. This phenomenon actually decreases in likelihood as port quantity increases. The extent of this decrease is the subject of present SW algorithm research.

Since OC-3 data rates are straightforward to achieve in memory for this design, consider the bandwidth requirement of OC-192 (9.9 Gbps) per input port. According to [9], modern embedded DRAM has a datapath of 256-bit, a latency of 11ns, and the page write time is 5ns. At 9.9 Gbps, there are 51ns to write one 512-bit packet every switch cycle. This means that even if two packets must be written to the same memory module in a switch cycle, there is 2 x (5+11) = 32ns required to accomplish this.

For a large switch, it is necessary to know how much memory will be needed to achieve acceptable performance. To find out how much memory is required for a given port quantity, it is necessary to run shared memory switch simulations under a practical set of assumptions. The traffic assumptions are based on the traffic model given in [1]. This traffic model is a two-state ON-OFF model with an average burst length (ABL) of eight. The Matlab file *b_traffic.m* is used for this simulation and can be found in the Appendix. The load is varied from 0.5 to 0.9 in increments of 0.1. The goal of this simulation is to find what the minimum packet memory size is for ten million packets. The reason this quantity of packets is chosen is that it correlates the $10^{-9}$ bit error rate of the network optical fiber [10]. This means that there is a greater chance of a packet being dropped because of a bit error than a packet being dropped by the switch for memory reasons. To explain this some simple

probability is used. The chance of a bit error is $10^{-9}$ [10], so the chance that a packet is dropped due to this bit error is:

$$10^{-9} \text{ x } 512 \cong \frac{1}{2000000}$$

This probability is much greater than the probability of a packet being dropped given the simulated memory requirement, which is at least $10^{-7}$.

The shared memory switch used in the simulations is a generic shared memory switch model that uses complete sharing of the memory space. This model is best explained by using a one-dimensional memory array, input ports, and output ports. Consider modeling a 4x4 switch with a packet memory of 12. A sample simulation is given in Table 4.3. The destination port number of the packets on the input is shown in column 2 as an array (zero represents no packet). The output packets are represented in an array in column 4. The switch memory is shown as an array in column 3.

Table 4.3 Shared-Memory Switch Simulation Example

| Switch Cycle | Packets at Input Ports 1 to 4 | Memory Array [1 2 3 4 5 6 7 8 9 10 11 12] | Packets at Output Ports 1 to 4 |
|---|---|---|---|
| 1 | [1 1 2 0] | [0 0 0 0 0 0 0 0 0 0 0 0] | [0 0 0 0] |
| 2 | [3 3 2 1] | [1 1 2 0 0 0 0 0 0 0 0 0] | [0 0 0 0] |
| 3 | [1 1 0 4] | [3 3 2 1 1 0 0 0 0 0 0 0] | [1 2 0 0] |
| 4 | [3 1 4 0] | [1 1 4 3 1 0 0 0 0 0 0 0] | [1 2 3 0] |

After adding the packets at the input, the simulator used in this design searches the memory array for port 1, port 2, port 3, and port 4. One packet may be removed for each port number every switch cycle. If there is no port number found for a given port, then nothing happens.

If there is no room to add packets in the array, then the array size is increased. At the end of the simulation (ten million packets), the array size is reported. This size is the minimum packet memory required for the switch not to drop one in ten million packets.

The simulation is written in Microsoft Visual Studio using C++. The code is given in the Appendix. The executable reads the traffic files generated by the Matlab file *packet_file.m* and runs the simulation based on this data.

The results of the memory simulation are shown in Figure 4.7 and in Table 4.4. Note that Figure 4.7 shows a non-linear port quantity on the X-axis.

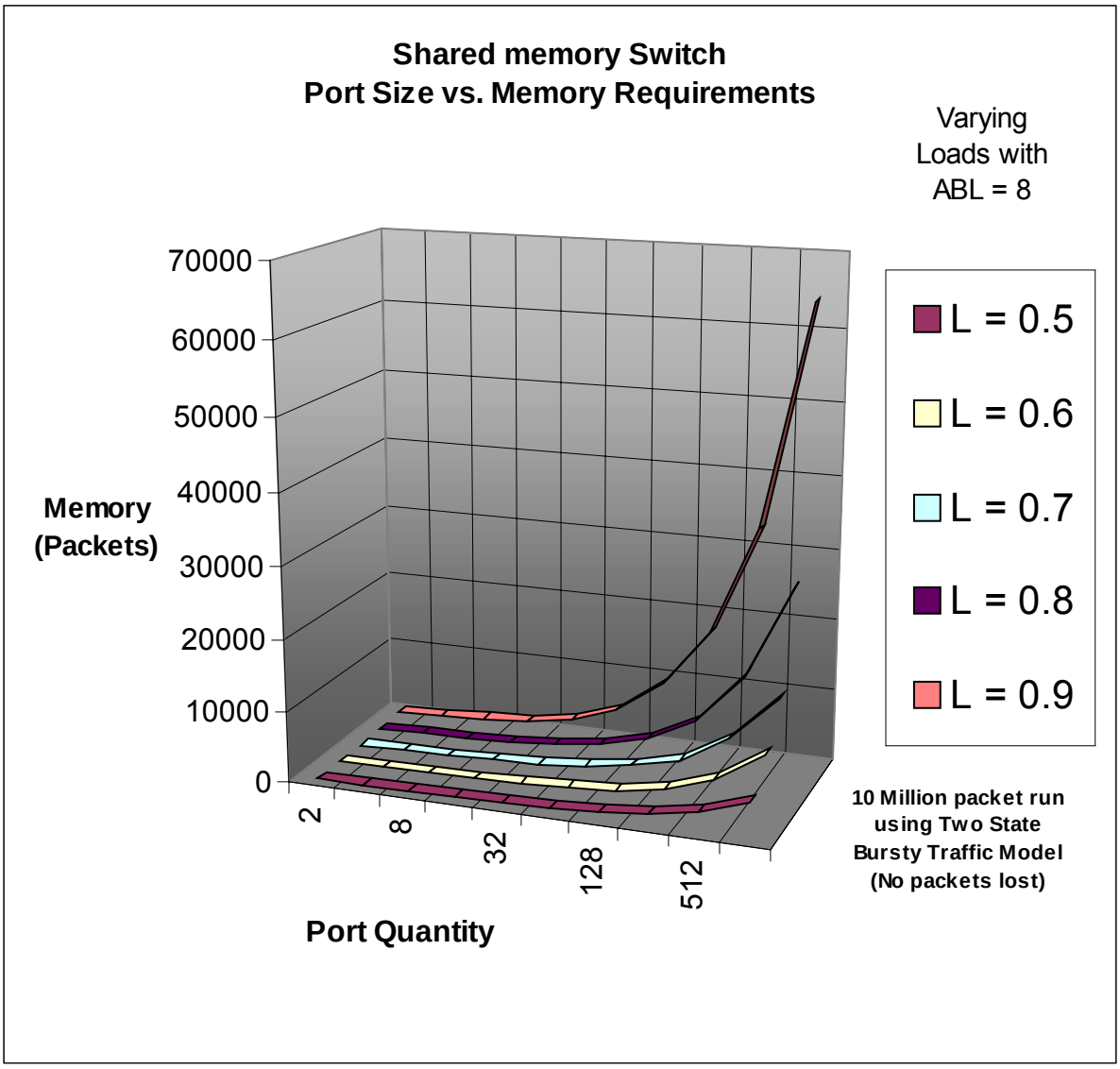



Figure 4.7 Port Quantity vs. Packet Memory Simulation For Various Traffic Loads

Table 4.4 Numerical Results of Simulations

| Port Quantity | Memory (packets) L = .5 | Memory (packets) L = .6 | Memory (packets) L = .7 | Memory (packets) L = .8 | Memory (packets) L = .9 |
|---|---|---|---|---|---|
| 2 | 128 | 128 | 128 | 192 | 384 |
| 4 | 128 | 192 | 256 | 384 | 832 |
| 8 | 192 | 256 | 320 | 512 | 1152 |
| 16 | 256 | 320 | 576 | 832 | 1600 |
| 32 | 320 | 512 | 768 | 1344 | 2688 |
| 64 | 576 | 960 | 1408 | 2240 | 4992 |
| 128 | 896 | 1408 | 2432 | 3904 | 9664 |
| 256 | 1536 | 2432 | 4096 | 7168 | 17728 |
| 512 | 2816 | 4672 | 8000 | 14144 | 33024 |
| 1024 | 4992 | 8896 | 14336 | 27392 | 64064 |

The results of this simulation yield an interesting relation between the port quantity and the memory size requirements. As the port quantity doubles, the memory size requirement does not quite double. This suggests a slightly logarithmic relation between port quantity and the required memory size.

Assuming a Load of 0.8, these results suggest that at least 27392 packets should be used in a 1024x1024 shared memory switch. This is based on the assumption that a load of 0.8 is representative of network traffic [1]. A packet quantity this large requires about 14 Mbits of RAM. According to [9], implementing this size in embedded DRAM is a practical solution. In other words this design does not require off-chip RAM, which would pose a considerable memory bandwidth problem. For smaller switches such as a 16x16, only 832 packets are required, which is about 425 kbits of RAM. This could be accomplished with embedded SRAM, which has a much higher data bandwidth than the embedded DRAM.

# Chapter 5

# Verification

To verify this packet switch hardware, a realistic traffic model is necessary. This model is contained in the Matlab file *b_traffic.m* that was obtained from [1]. It is the same model used in Chapter 4 for the memory simulations, which is based on a two-state ON-OFF model. The traffic used for this verification has an average burst length (ABL) of eight, a load of 0.999, and consists of approximately one million packets.

The Software version of the Sliding-Window Switching Algorithm is used for verifying the hardware and is run by the top-level file *test_sw.m*. It is contained in the Algorithm files listed in the Appendix. This algorithm, written in Matlab, is used to generate two files based on the input traffic created by *b_traffic.m*:

1. *verilog_port.dat* – the traffic file for the Verilog simulation.
2. *test_output.dat* – the software algorithm output, listed as the *i, j, k, d* parameters of the packets at the output port.

The *test_output.dat* file is in the same format as the Verilog output file (discussed below) so that the comparison is simple. The traffic file consists of a sequential list of destination port numbers. Each group of four port numbers is treated as one switch cycle:

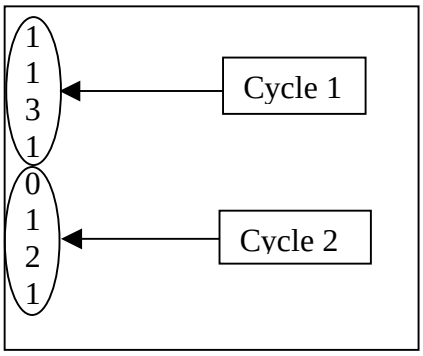



Figure 5.1 Example port data output in *verilog_port.dat*

The Verilog test fixture *test_full5.v* reads this file (after it is loaded into memory) four port numbers at a time. These port numbers are then injected into the packet data, which is stored statically in the test fixture, to create a 512-bit packet. Note that the port number is stored as the first three bits of each packet. The packet is then serialized and "fed" into the PAC. This happens simultaneously at all four of the PAC input ports.

After a few delay switch cycles, the hardware simulation starts streaming out packet bits serially. These packets now have the *i, j, k,* and *d* parameters in the first 13 bits. The packets are parallelized in the test fixture *test_full5.v*, so that the data may be extracted, mainly the *i, j, k,* and *d* parameters. These parameters are written to the file *verilog_test_output.dat* in the format *i/j/k/d.* The Matlab code, *test_sw.m*, creates this same format. Here is an example of the output of both the Matlab and the Verilog output file:

```
0/0/0/0          // Port 1
0/0/0/0          // Port 2
6/ 1/1/3         // Port 3
1/ 1/1/4         // Port 4

5/ 2/1/1
0/0/0/0
3/ 2/1/3
2/ 2/1/4

4/ 3/1/1
0/0/0/0
5/ 3/1/3
2/ 3/1/4
```

Figure 5.2 Output example. *verilog_test_output.dat* and *test_output.dat* follow this format

To run this simulation using the *Makefile*, the command "make test" is used. This *Makefile* contains all the compiler directives to get the FPGA simulation models to compile. Note that for one million packets, this simulation can take one to two days. This requires the use of either the Load Sharing Facility or a fast Linux machine. This simulation was run using bock.ece.ncsu.edu and required only 14 hours.

The output files from both simulations contain only the packet parameters used for assignment as discussed above. These parameters are obtained from the packet on the output port. After the simulation is completed, the two files are compared using the UNIX command:

```
diff –w verilog_test_output.dat test_output.dat
```

If all the entries are the same and both files are the same length, then nothing is returned.

## 5.1 Results

After debugging the test fixture to correctly run one million packets, no output from the diff command was achieved.

As discussed in Chapter 3, there is a rare case of duplicate *i* parameter assignments that must be considered. The test fixture is configured to detect when this happens and output to a file. The purpose of this detection is to make sure it is actually happening. If it is found that this duplicate *i* parameter assignment is happening and that the simulation outputs match up, then it can be safely assumed that the hardware is handling this case properly. The file that stores the record of the duplicate *i* assignments is *verilog_d_i.dat.* After the one-million packet simulation this file contained 138 accounts of duplicate *i* assignment. Since the simulation output files *verilog_test_output.dat* and *test_output.dat* are both identical, this case of duplicate *i's* is working properly in hardware.

# Chapter 6

# Conclusions and Future Work

This thesis work implemented and verified a 4x4 Sliding-Window Packet Switch in hardware using Verilog, a hardware description language. The hardware design is then analyzed to uncover scalability constraints of the Sliding-Window packet switch algorithm. The scalability constraints that are discussed in this thesis are the Parameter Assignment Circuit (PAC) Processors, the Input Interconnection Network (IIN) and the Output Interconnection Network (OIN), and the Memory Modules.

The primary scalability timing constraints are in the PAC Processors. Since these processors must assign all the parameters packet by packet in a given switch cycle, any critical paths that can be optimized will speed up the PAC significantly. One critical path uncovered in this work is in Processor 1 (which loops through Processor 2), which is the determination of whether or not a packet is dropped. This has many dependencies, such as $j\_f$ and $ES\_j$. Fortunately, only one small part of this critical path grows with port quantity. This growth is only logarithmic as discussed in Chapter 4. This critical path also scales better for large port quantity than the $i$ search in Processor 2.

As port quantity increases (N), the $i$ search in Processor 2 that must be accomplished grows at a rate of 2N. This search is accomplished using a priority encoder as discussed in Chapter 3. Although a synthesized priority encoder may be parallelized to an extent, it poses a serious bottleneck for the design for large port quantity. One solution is to use a CMOS priority encoder designed using Three Level Folding and Three Level Look Ahead (TLF-TLLA) as proposed in [7]. This design would nearly eliminate this $i$ search bottleneck by

cutting the logic chain for a 1024-bit PE (used in a 512x512 switch) from 66 to 10 gates. One tradeoff in doing this is cost. This would be much more expensive and difficult to accomplish, but the more than six-fold speedup of this method may be worth it for large port quantity switches.

The IIN and OIN are based on the same hardware design using a crossbar. As port quantity grows, the use of combinational logic to implement the crossbar becomes less practical. For a 512x512 switch, the use of a VLSI crossbar such as the one developed in [8] would be more practical than using large amounts of combinational logic. This method has the best throughput, but is also the most expensive to implement. Another common and simple crossbar implementation is to use tri-state buffers. This method would likely be the simplest if the relatively poor data throughput is not a factor, such as at OC-3 data rates.

The Memory Modules distribute memory bandwidth between physically separate RAMs. This avoids the memory bandwidth bottleneck present in other packet switch architectures. The individual Memory Module bandwidth can be viewed to be the same as the bandwidth at the input port. Even if embedded DRAM is used for this memory instead of SRAM, there will not be any considerable bandwidth bottlenecks. As port quantity is increased, the memory bandwidth stays the same. The required amount of packet memory does not.

The relationship of packet memory to port quantity is simulated and analyzed in Chapter 4. The results of these simulations show that for large port quantities such as a 1024x1024 switch, approximately 14 Mbits of embedded DRAM would be required. This is practical with modern embedded DRAM. Therefore going off-chip for packet storage would

not be necessary. For these reasons memory bandwidth would not be a problem regardless of port quantity.

## 6.1 Open Issues and Future Work

The Parameter Assignment Circuit has two main critical paths as discussed in Chapter 4. It may be practical to break the critical path in Processor 1 into separate pipeline cycles. The practicality of pipelining Processor 1 may be the subject of future research. Breaking up this critical path would speed up Processor 1 considerably.

The critical path in Processor 2, the *i* assignment search, would benefit greatly by using the CMOS TLF-TLLA priority encoder. This also raises a question of whether or not this is a cost-effective solution. The gate delay improvement from using this priority encoder design would almost eliminate this critical path in this processor. Eliminating a critical path may be worth the cost of a custom CMOS VLSI.

The Input and Output Interconnection Networks both rely heavily on a large crossbar. The use of tri-state buffers to accomplish this could be explored further. One aspect to consider is what the path delay of this implementation would be, or what data throughput is achievable per line. Also what area requirements for large amounts of ports could be compared to other methods.

In addition, the use of VLSI crossbars such as the one given in [8] needs research. For example, what would the effect on throughput be on creating a larger crossbar than 128x128 from the 8x8 cells given in [8]? The cost of implementing this design must be considered. Are the improvements worth the added cost of using this custom VLSI design as opposed to tri-state buffers from a cell library?

The embedded DRAM referenced for this design is IBM IP that is currently only available in 1Mbit blocks [9]. Many smaller blocks of embedded DRAM would be more practical to this design because of the memory architecture of this algorithm. Custom embedded DRAM may need to be designed if smaller blocks are not commercially available.

# References

[1] S. Kumar, “The Sliding-Window Packet Switch: A New Class of Packet Switch Architecture With Plural Memory Modules and Decentralized Control” in *IEEE Journal on Selected Areas in Communications*, vol. 21, pp.656-673, May 2003.

[2] T. Kozaki, N. Endo, Y. Sakurai, O. Matsubara, M. Mizukami, K. Asano “32 X 32 Shared Buffer Type ATM Switch VLSI’s for B-ISDN’s” in *IEEE Journal on Selected Areas in Communications*, vol. 9, pp.1239-1247, October 1991.

[3] K. Yamanaka et al. “Scalable shared-buffering ATM switch with a versatile searchable queue,” in *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 773-784. June 1997

[4] F. A. Tobagi, “Fast packet switch architectures for broadband integrated services digital networks,” *Proceedings of the IEEE*, vol. 78, pp. 133–167, Jan. 1990.

[5] D. E. Culler and J.P. Singh, Parallel Computer Architecture, Morgan Kaufmann Publishers, 1999. ISBN 1-55860-343-3

[6] D. J. Smith, “HDL Chip Design,” Doone Publications, 1996, ISBN 0-9651934-3-8

[7] C. Huang, J. Wang and Y. Huang “Design of High-Performance CMOS Priority Encoders and Incrementer/Decrementers Using Multilevel Lookahead and Multilevel Folding Techniques,” IEEE Journal of Solid-State Circuits, vol. 37, no. 1, January 2002, pp 63-76.

[8] J. Damiano et al., “A High-Speed & High-Capacity Single-Chip Copper Crossbar” http://www4.ncsu.edu/~jdamian/copper.html

[9] IBM ASIC Products Datasheet; Cu-11 Embedded DRAM, September 2003, http://ibm.com/chips

[10] Fiber Optics Fact Sheet, http://www.vlab.org/optics_fact_sheet.doc

[11] S. Kumar and T. Doganer, "Memory-Bandwidth Performance of the Sliding-Window based Routers/Switches," *IEEE Conference on Local Area and Metropolitan Area Networks*, San Francisco, CA, April 2004.

[12] S. Kumar, T. Doganer and A. Munoz, "Effect of Traffic Burstiness on Memory-Bandwidth of the Sliding-Window Switch Architecture," International Conference on Networking, French Caribbean, March 2004.

# Appendix

## Verilog Switch Hardware

```
/*-----------------------------------------------------------*/
/*
(c) 2004 B.R. Phelps, North Carolina State University
Module PAC_600.v

Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      This uses the SW_Counter module & tables such as LC_j,
      LC_k, and Qd to assign the j and k values according to the d
      value given.

      One aspect of the algorithm for Processor 600 (ES_j_in)
      requires information from the Scan Table which is in PAC_650.
      Since PAC_650 is one cycle behind this module, PAC_650 uses
      predictive logic based on the current j(unassigned) logic
      value from 600 to assign ES_j.

*/
/*-----------------------------------------------------------*/
//`include "~/MSThesis/Project_Verilog/PAC/PAC_SWCounter.v"
module PAC_600(clock, reset,
            start_600_in, SW_enable, d_ctrl_in,
            d1_in, d2_in, d3_in, d4_in, ES_j_in,
            pd_525, pd_508,
            j_524_out, jd, kd, d_out);
/*-----------------Port Declarations----------------------*/
input clock, reset;
input start_600_in, SW_enable;
input[2:0] d1_in, d2_in, d3_in, d4_in;
input[1:0] d_ctrl_in;
input[3:0] ES_j_in;

output pd_525, pd_508;
output[1:0] kd;
output[3:0] jd;
output[3:0] j_524_out;
output[2:0] d_out;

/*-----------------------------------------------------------*/

parameter SIGMA = 12, P=2, P_DOT_SIGMA=7'd24, HEADER1_BIT=8,
HEADER2_BIT=12, NxNm1=3;
parameter NxN=4;
/*-----------------Wire & Reg's---------------------------*/
integer k;
```

```
reg [1:0]LC_k[0:NxN];
reg [6:0] Qd [0:NxN];
reg [3:0]LC_j[0:NxN];

reg [2:0] d, d_out;
reg [3:0]jd;
wire[3:0] j_f, LC_j_p1;
reg [1:0] kd;
wire[1:0]k_f, k_mo, LC_k_p1;

wire enable, pd_525, pd_508, valid_packet ;
wire[1:0] SW_k,  k_out;
wire[3:0] SW_j, j_out;
wire[6:0] Qd_t, Qd_f, Qd_ff;

wire[2:0] r;
wire pd_ES_j, DC1;
/*-------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
assign Qd_t = (Qd[d] + |d);   // add !=0

assign pd_508 = (Qd_t > P_DOT_SIGMA);
assign Qd_f = (pd_508||pd_525) ? Qd[d] : Qd_t; // mux1 (Subtracts a packet
after the initial add)

assign LC_k_p1 = (LC_k[d]==P)? 2'd1 : LC_k[d] + 1; // increment LC_k in
mod fashion
assign LC_j_p1 = (LC_j[d]==SIGMA)? 4'd1 : LC_j[d] + 1; // increment LC_j
in mod fashion

assign k_mo = (LC_j_p1 == 1)? LC_k_p1: LC_k[d]; // mux2

assign j_f = (Qd_t==1)? SW_j : LC_j_p1;  // mux3a
assign k_f = (Qd_t==1)? SW_k : k_mo;     // mux3b

/*------------------Assigns for 524-525 --------------------*/
// some of this logic is redundant (unnecessary) to stay true to the
algorithm
assign j_524_out = (d==0)?0:j_f;  // don't output anything if d is 0
assign r = (((SIGMA > (Qd[1]+(d==3'd1))) + (SIGMA > (Qd[2]+(d==3'd2)))) +
((SIGMA > (Qd[3]+(d==3'd3))) + (SIGMA > (Qd[4]+(d==3'd4))))); // add one
to the total to include current packet
assign pd_ES_j = (Qd_t<SIGMA) && (~DC1) && (d>0)&&(~(ES_j_in > r));
assign DC1 = P_DOT_SIGMA < Qd_t;                // Drop Case 1
assign pd_525 = pd_ES_j;

assign valid_packet = (|d)&&(!pd_525)&&(!pd_508);  // if d is >0 and
packet is not

                              // dropped then the packet is valid
/*----------Counter update Signals------------------------*/
assign enable = SW_enable;

/*- - - - - - - - - - - - - - - - -  - -  - - - - - - - - -*/

/*-------------------------------------------------------*/
```

```
/*-----------------Combinational logic --------------------*/

// This is mux0
always @(d_ctrl_in or d1_in or d2_in or d3_in or d4_in)
      begin
            casex(d_ctrl_in)
                  2'b00 : d = d1_in;
                  2'b01 : d = d2_in;
                  2'b10 : d = d3_in;
                  2'b11 : d = d4_in;
                  default: d = d1_in;
            endcase
      end


/*-------------------------------------------------------*/


/*-----------------Flip Flops-----------------------------*/
always@(posedge clock)
      begin
            if (!reset)
                  begin
                        jd<=0; kd<=0; d_out<=0;
                        for(k=0; k <= NxN; k = k + 1)
                              begin
                                    LC_j[k]<=1; LC_k[k]<=1; Qd[k]<=0;
                              end
                  end
            else if (start_600_in)
                  begin
                        Qd[d] <= Qd_f;
                        if (!pd_508)  // these are not updated unless 508
is low
                              begin
                                    LC_j[d] <= j_f; LC_k[d] <= k_f;
                              end

                        if (valid_packet)
                              begin
                                    jd <= j_f; kd <= k_f; d_out <= d;
                              end
                        else
                              begin
                                    jd <= 0; kd <= 0; d_out <= 0;
                              end
                  end

            else if (SW_enable)     // Counter update signal
                  begin
                        jd<=0; kd<=0; d_out<=0; //reset these for clarity

                        Qd[1] <= (Qd[1]==0)?0:Qd[1]-1'b1;
                        Qd[2] <= (Qd[2]==0)?0:Qd[2]-1'b1;
                        Qd[3] <= (Qd[3]==0)?0:Qd[3]-1'b1;
                        Qd[4] <= (Qd[4]==0)?0:Qd[4]-1'b1;
```

```
                  end
            else
                  d_out <= 0; // no packet signal is default

      end
/*-------------------------------------------------------*/


/*-----------------Module Declarations--------------------*/
PAC_SWCounter SW_Count(clock, reset, enable, SW_j, SW_k);

/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
(c) 2004 B.R. Phelps, North Carolina State University
Module PAC_600.v

Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Uses j to index the Scan Table.  The ST(j) vector is then
      searched (along with temp_i)using an 8 bit priority encoder.
      The index + 1 of the first slot found is the assigned i value.

      Logic is also contained here for finding ES_j based on the
      Scan Table and the unassigned j value.

*/
/*-------------------------------------------------------*/
//`include "PAC_SWCounter.v"
module PAC_650(clock, reset,
            start_650_in, SW_enable,
            j_524_in, j_in, k_in, d_in,
            i_out, j_out, k_out, d_out,
            ES_j, pd_i_full_out);
/*-----------------Port Declarations----------------------*/
input clock, reset;
input start_650_in, SW_enable;

input [1:0] k_in;
input [2:0] d_in;
input [3:0] j_524_in, j_in;

output pd_i_full_out;
output [1:0] k_out;
```

```
output[3:0] i_out;
output[2:0] d_out;
output[3:0] j_out;
output [3:0] ES_j;
/*-------------------------------------------------------*/

parameter SIGMA = 12, P=2, P_DOT_SIGMA=24, HEADER1_BIT=8, HEADER2_BIT=12,
NxNm1=3;
parameter NxN=4;
/*-----------------Wire & Reg's--------------------------*/
integer k1,k2,k3;

reg [1:0] ST [0:95];
reg [7:0] temp_i;
reg [7:0] temp_if;

reg [2:0] d_out;
reg [3:0] j_out;
reg [1:0] k_out;
reg [3:0] i_out, i_addr1, i_addr2;
reg pd_i_full_out;

wire enable;
wire[1:0] SW_k;
wire[3:0] i_addrf;
wire[3:0] SW_j;
wire[3:0] SW_j_addr;
wire[3:0] ES_j;
wire[3:0] ES_temp1, ES_temp2;
wire[3:0] j_524_addr, j_addr;
wire[7:0] ST_j;
reg[7:0] ST_j_f;
/*-------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
assign j_addr = (j_in==0)?4'b0:j_in-1'b1;  // hardware starts at 0
//assign j_addr = j_in-1'b1; // Hardware starts at 0
assign SW_j_addr = SW_j-1'b1; // Hardware starts at 0

assign ST_j[0] = |ST[{j_addr,3'd0}];      // check ST_(j,i)==0
assign ST_j[1] = |ST[{j_addr,3'd1}];      // finds all empty i slots in
current ST(j)
assign ST_j[2] = |ST[{j_addr,3'd2}];
assign ST_j[3] = |ST[{j_addr,3'd3}];
assign ST_j[4] = |ST[{j_addr,3'd4}];
assign ST_j[5] = |ST[{j_addr,3'd5}];
assign ST_j[6] = |ST[{j_addr,3'd6}];
assign ST_j[7] = |ST[{j_addr,3'd7}];

assign j_524_addr = (j_524_in==0)?4'b0:j_524_in - 1'b1;  // hardware
starts at 0
//assign j_524_addr = j_524_in - 1'b1;  // hardware starts at 0

// here speculate what ES_j will be in the next cycle since ST_j has not
been updated
// for packet on j_in, k_in, d_in => ( (j_524_in == j_in) + ...)
```

```
assign ES_j = (ES_temp2 > 4'd8)?4'd8:ES_temp2;

// mux1
assign ES_temp2 = (j_524_in==0)?8:ES_temp1-((j_524_in ==
j_in)&&(ES_temp1>0));  //Subtract 1 if the 2 j values's are equal

//adder1
assign ES_temp1 = (   ((ST[{j_524_addr,3'd0}]==0) +
(ST[{j_524_addr,3'd1}]==0)) +
                              ((ST[{j_524_addr,3'd2}]==0) +
(ST[{j_524_addr,3'd3}]==0))   ) +
                              (   ((ST[{j_524_addr,3'd4}]==0) +
(ST[{j_524_addr,3'd5}]==0)) +
                              ((ST[{j_524_addr,3'd6}]==0) +
(ST[{j_524_addr,3'd7}]==0))   ) ;  // Add up the # of empty slots

// mux_i
assign i_addrf = (i_addr1 == 4'd8) ? i_addr2 : i_addr1;
/*-------------------------------------------------------*/

/*----------Counter update Signals------------------------*/
assign enable = SW_enable;


/*- - - - - - - - - - - - - - - - -  - -  - - - - - - - - -*/

/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/

// Must flip both vectors to work properly
always@(ST_j)
      begin
            for (k2=0; k2<8; k2 = k2+1)
                  begin
                        ST_j_f[k2]=ST_j[7-k2];
                  end
      end

always@(temp_i)
      begin
            for (k3=0; k3<8; k3 = k3+1)
                  begin
                        temp_if[k3]=temp_i[7-k3];
                  end
      end

// PE1
always@(ST_j_f or temp_if)
      begin
            casex(ST_j_f|temp_if)  // bitwise OR of the two vectors
                  8'b0xxxxxxx : i_addr1 = 4'd0;
                  8'b10xxxxxx : i_addr1 = 4'd1;
                  8'b110xxxxx : i_addr1 = 4'd2;
                  8'b1110xxxx : i_addr1 = 4'd3;
                  8'b11110xxx : i_addr1 = 4'd4;
                  8'b111110xx : i_addr1 = 4'd5;
```

```
                  8'b1111110x : i_addr1 = 4'd6;
                  8'b11111110 : i_addr1 = 4'd7;
                  //8'b11111111 : i_addr1 = 4'd8;
                  default : i_addr1 = 4'd8;
            endcase
      end

// if the othe i search fails (i_addr1==8)then use this search:
// PE2
always@(ST_j_f)
      begin
            casex(ST_j_f)
                  8'b0xxxxxxx : i_addr2 = 4'd0;
                  8'b10xxxxxx : i_addr2 = 4'd1;
                  8'b110xxxxx : i_addr2 = 4'd2;
                  8'b1110xxxx : i_addr2 = 4'd3;
                  8'b11110xxx : i_addr2 = 4'd4;
                  8'b111110xx : i_addr2 = 4'd5;
                  8'b1111110x : i_addr2 = 4'd6;
                  8'b11111110 : i_addr2 = 4'd7;
                  //8'b11111111 : i_addr2 = 4'd8;
                  default : i_addr2 = 4'd8;
            endcase
      end
/*-------------------------------------------------------*/


/*-----------------Flip Flops----------------------------*/
always@(posedge clock)
      begin
            if (!reset)
                  begin
                        temp_i <= 0;
                        d_out <= 0; j_out <= 0; k_out <= 0; i_out <=0;
                        pd_i_full_out <= 0;
                        for(k1=0; k1 <= 95; k1 = k1+ 1)
                              begin
                                    ST[k1] <= 0;
                              end
                  end
            else if (start_650_in) // start signal has priority over
SW_enable
                  begin
                        if ((i_addrf != 4'd8)&&(|d_in))
                              begin
                                    i_out <= i_addrf + 1'b1;
                                    k_out <= k_in;
                                    j_out <= j_in;
                                    d_out <= d_in;
                                    temp_i[i_addrf[2:0]] <= 1;

                                    ST[{j_addr, i_addrf[2:0]}] <= k_in;
// Update ST with new k
                                    pd_i_full_out <= 0;
                              end

                        else  // ST_j is full or no packet is present
```

```
                              begin
                                    d_out <= 3'd0; j_out <= 0; k_out <= 0;
i_out <=0;
                                    pd_i_full_out <= (i_addrf == 4'd8);
                              end

                  end
            else if (SW_enable) // ST update (clears appropriate data)
                  begin
                        temp_i<=8'd0;
                        d_out <= 0; j_out <= 0; k_out <= 0; i_out <=0;
                        pd_i_full_out <= 0;
                        ST[{SW_j_addr, 3'd0}] <= (ST[{SW_j_addr,
3'd0}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd0}];
                        ST[{SW_j_addr, 3'd1}] <= (ST[{SW_j_addr,
3'd1}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd1}];
                        ST[{SW_j_addr, 3'd2}] <= (ST[{SW_j_addr,
3'd2}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd2}];
                        ST[{SW_j_addr, 3'd3}] <= (ST[{SW_j_addr,
3'd3}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd3}];
                        ST[{SW_j_addr, 3'd4}] <= (ST[{SW_j_addr,
3'd4}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd4}];
                        ST[{SW_j_addr, 3'd5}] <= (ST[{SW_j_addr,
3'd5}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd5}];
                        ST[{SW_j_addr, 3'd6}] <= (ST[{SW_j_addr,
3'd6}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd6}];
                        ST[{SW_j_addr, 3'd7}] <= (ST[{SW_j_addr,
3'd7}]==SW_k)?2'd0:ST[{SW_j_addr, 3'd7}];
                  end
      end
/*-------------------------------------------------------------*/


/*-----------------Module Declarations---------------------*/
// This is used only to know what to clear in the Scan Table
PAC_SWCounter SW_Count_650(clock, reset, enable, SW_j, SW_k);
/*-------------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------------*/
/*
Module PAC_Header.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Reads in data serially, and parallelizes it to 64 bit words
      Uses control signals from the NCFSM to record the
      d (destination) from the packet header.  The packet header
      is assumed to be 13 bits for this project, and is the 1st 13 bits
```

```
      that arrive (the 13 LSB's).  The d is sent to the PAC_600
      module for parameter assignment.

*/
/*-------------------------------------------------------*/

//`include "../Mega_Functions/shift_reg2.v"
module PAC_Header(clock, reset,
                        ser_in,     /* serial data input*/
                        d_valid_in, /* d is now valid*/
                        sp_out,                 /* serial-parallel out */
                        d_out);           /* d extracted */
/*-----------------Port Declarations----------------------*/
input clock, reset;
input ser_in, d_valid_in;

output[63:0] sp_out;
output[2:0] d_out;


/*-------------------------------------------------------*/

/*-----------------Wire & Reg's---------------------------*/
wire[63:0] sp_out;

reg[2:0] d_out;


parameter SIGMA = 12, P=2, HEADER1_BIT=8, HEADER2_BIT=12, NxNm1=3;
/*-------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/

/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
/*-------------------------------------------------------*/

/*-----------------Flip Flops-----------------------------*/


always@(posedge clock)
      begin
            if(!reset)
                  d_out <= 0;
            else if(d_valid_in)     // record the d port value
                  begin                         // while it is still valid
                                                      // (from 1st 3 bits
of packet)
                        d_out <= sp_out[63:61];
                  end
      end
/*-------------------------------------------------------*/


/*-----------------Module Declarations--------------------*/
```

```
//    Quartus II shift register Megafunction
//    used for parallelizing the serial data
//          This is synthesized as well
shift_reg2 shift1(.clock(clock), .sclr(!reset),
  .shiftin(ser_in),
      .q(sp_out) );
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module PAC_Memory.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Reads in data serially, and parallelizes it to 64 bit words
      Uses control signals from the NCFSM to record the
      i,j,k,d values from the PAC_650.  The i,j,k,d values are
      muxed into the packet header to the RAM at the time signaled
      by NCFSM.

      The RAM is used to create a pipeline delay of one
      switch cycle.  This is done to stay consistent with the
      algorithm.

      The packet header is assumed to be 13 bits for this project,
      and is the 1st 13 bits that arrive (the 13 LSB's).  The d is
      sent to the PAC_600 module for parameter assignment.

*/
/*-------------------------------------------------------*/

//`include "../Mega_Functions/ram_pac.v"
module PAC_Memory(clock, reset,                                  /* ctrl
signals*/
                        addr_in, wren1_in, ctrl1_in, ijkd_valid_in,
                        sp_in, /* serial-parallelled data input*/
                        d_in, j_in, k_in, i_in,       /* Output of 600-650
pipeline */
                        sp_out);    /* PAC_output*/
/*-----------------Port Declarations----------------------*/
input clock, reset;

input[63:0] sp_in;

input  wren1_in, ctrl1_in, ijkd_valid_in;
input[2:0] addr_in;

input[2:0] d_in;
```

```
input[3:0] i_in;
input[3:0] j_in;
input[1:0] k_in;

output[63:0] sp_out;


/*-------------------------------------------------------*/

/*-----------------Wire & Reg's--------------------------*/
parameter sigma = 12, p = 2, HEADER1_BIT=8, HEADER2_BIT=12;

// The 13 LSB bits of the packet contains the Header:
//H2    12 11  10  9  H1   8   7  6  5   4   3    2  1 0
//      [   i      ]      [   j      ]  [  k ]   [  d  ]

reg[2:0] dd;
reg[3:0] id;
reg[3:0] jd;
reg[1:0] kd;
reg[2:0] d;
reg[3:0] i;
reg[3:0] j;
reg[1:0] k;

wire[2:0] addr_p1;

wire[63:0] ram1_out;
wire[HEADER2_BIT:0] mux1_out;

wire[63:0] sp_out;

/*-------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
// Create read address from write address, read address is
//    the write address plus 1
// This creates the neccessary PAC pipeline delay
assign addr_p1 = (addr_in==3'd7)?3'b000:addr_in + 1'b1;

// Header routing circuitry; injects i,j,k,d into the packet
//          at the right time (signal sent from the FSM)
assign mux1_out = (ctrl1_in)?{i, j, k, d}:sp_in[HEADER2_BIT:0];  // mux1

/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
/*-------------------------------------------------------*/

/*-----------------Flip Flops-----------------------------*/
always@(posedge clock)
      begin
            if(!reset)
                  begin
                        i <= 0; j <= 0; k <= 0; d <= 0;
                  end
            else if(ijkd_valid_in)  // Record i,j,k,d from PAC_650
```

```
                  begin
                        i <= i_in; j <= j_in; k <= k_in; d <= d_in;
                  end
      end
/*-------------------------------------------------------*/


/*-----------------Module Declarations--------------------*/
//    Quartus II RAM Megafunction
//    used for the PAC pipeline delay (stores one packet)
//          This is synthesized
ram_pac ram1(           //(8 cycle delay or 1 switch cycle)
      .data({sp_in[63:(HEADER2_BIT+1)],mux1_out}),
      .wraddress(addr_in),
      .rdaddress(addr_p1),
      .wren(wren1_in),
      .inclock(clock),
      .outclock(clock),
      .aclr(!reset),
      .q(sp_out));


/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module PAC_NCFSM.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      This is simply a counter that sends out control signals at
      specific times (values).  These control signals keep all the
      parts of the PAC in sync.

      This module also creates a RAM address for the PAC_Memory modules

*/
/*-------------------------------------------------------*/

module PAC_NCFSM(clock, reset,                                    /* ctrl
signals*/
                        start_600_out, start_650_out, SW_enable_600_out,
SW_enable_650_out,
                         d_valid_out, d_ctrl_out,/* d in Header is now
valid*/
                        addr_out, ctrl1_out, ijkd_valid_out,
                        wren1_out);       /* d extracted */
/*-----------------Port Declarations----------------------*/
```

```
input clock, reset;
output d_valid_out, ctrl1_out, wren1_out;

output SW_enable_650_out, SW_enable_600_out, start_600_out, start_650_out;
output[2:0] addr_out;
output[3:0] ijkd_valid_out;
output[1:0] d_ctrl_out;

/*-------------------------------------------------------*/

/*-----------------Wire & Reg's--------------------------*/
wire d_valid_out, ctrl1_out, wren1_out;


reg[3:0] ijkd_valid_out; // one hot valid signal to each Memory pipeline
wire[2:0] addr_out;

reg[1:0] d_ctrl_out;
reg[8:0] counter;

reg start_600_out, SW_enable_600_out, start_650_out, SW_enable_650_out;

parameter SIGMA = 12, P=2, HEADER1_BIT=8, HEADER2_BIT=12;
/*-------------------------------------------------------*/

/*-----------------Assigns-------------------------------*/
assign addr_out = counter[8:6];

// d is now valid so record it
assign d_valid_out = (counter == 9'd3);

// Muxes the Header1 bits (i, j,k,d) for Ram to record them in mem
//assign ctrl1_out = ((counter > 9'd9) && (counter <= 9'd41));
assign ctrl1_out = ((counter == 9'h040));

// Ram1 write signal
assign wren1_out = (counter[5:0]==6'd0);



/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
// Muxes the Header2 bits  (i,j,k,d)

always@(counter)
begin
      case(counter)
            9'd6:ijkd_valid_out=4'b0001;
            9'd7:ijkd_valid_out=4'b0010;
            9'd8:ijkd_valid_out=4'b0100;
            9'd9:ijkd_valid_out=4'b1000;
            default: ijkd_valid_out=4'b0000;
      endcase
end

always@(counter)
```

```
begin
      case(counter)
            9'd4:start_600_out=1;
            9'd5:start_600_out=1;
            9'd6:start_600_out=1;
            9'd7:start_600_out=1;
            9'd8:start_600_out=0;
            default: start_600_out = 1'b0;
      endcase
end

always@(counter)
begin
      case(counter)
            9'd5:start_650_out=1;
            9'd6:start_650_out=1;
            9'd7:start_650_out=1;
            9'd8:start_650_out=1;
            9'd9:start_650_out=0;
            default: start_650_out = 1'b0;
      endcase
end

always@(counter)
begin
      case(counter)
            9'd8:SW_enable_600_out=1;
            default: SW_enable_600_out = 1'b0;
      endcase
end

always@(counter)
begin
      case(counter)
            9'd9:SW_enable_650_out=1;
            default: SW_enable_650_out = 1'b0;
      endcase
end

always@(counter)
begin
      case(counter)
            // These control signals mux the 4 d
            // values coming into PAC_600 at the correct times.
            9'd5:d_ctrl_out=2'b01;
            9'd6:d_ctrl_out=2'b10;
            9'd7:d_ctrl_out=2'b11;
            9'd8:d_ctrl_out=2'b11;
            default: d_ctrl_out = 2'b00;
      endcase
end
/*-------------------------------------------------------*/

/*-----------------Flip Flops----------------------------*/

always@(posedge clock)
      begin
```

```
            if(!reset)
                  counter <= 0;
            else
                  begin
                        counter <= counter + 1'b1;  // 512 cycle counter
for controlling

       // PAC tasks
                  end
      end
/*-------------------------------------------------------------*/

endmodule
```

---

```
(c) 2004 B.R. Phelps, North Carolina State University

module PAC_SWCounter(clock, reset, enable, SW_j, SW_k);           /*
outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset, enable;

output[3:0] SW_j;
output [1:0]SW_k;

/*-------------------------------------------------------------*/

/*-----------------Wire & Reg's---------------------------*/

wire[3:0] SW_j_add;
reg[3:0] SW_j;
reg [1:0]SW_k;

parameter SIGMA = 12, P=2;

/*-------------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/

assign SW_j_add= (SW_j==SIGMA)?4'd1:SW_j + 1'b1;  // SteP 410

/*-------------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
/*-------------------------------------------------------------*/

/*-----------------FliP FloPs-----------------------------*/

always@(posedge clock)
begin

      if(!reset)
            begin
                  SW_j <= 4'b0001; SW_k <= 2'b01;
            end
      else if (enable)
      begin
```

```
                  SW_j <= SW_j_add;// (SW_k mod p) ++
            if (SW_j_add == 4'b0001)
                  if (SW_k == P)  // could also be SW_k =~ SW_k;
                        SW_k <= 2'b01;
                  else
                        SW_k <= 2'b10;  // (SW_k mod P) ++
            else
                  SW_k <= SW_k;
      end


end

/*-------------------------------------------------------------*/


/*-----------------Module Declarations---------------------*/
/*-------------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------------*/
/*
Module PAC_Full.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Connects all the lower level PAC modules together to form
      one 4x4 PAC.

*/
/*-------------------------------------------------------------*/
module PAC_Full(clock, reset, ser_in, sp_out1, sp_out2, sp_out3, sp_out4);
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[3:0] ser_in;
output[63:0] sp_out1, sp_out2, sp_out3, sp_out4;

/*-------------------------------------------------------------*/

parameter SIGMA = 12, P=2, P_DOT_SIGMA=24, HEADER1_BIT=8, HEADER2_BIT=12,
NxNm1=3;
parameter NxN=4;

/*-----------------Wire & Reg's---------------------------*/
wire[63:0] Mem1_sp_out, Mem2_sp_out, Mem3_sp_out, Mem4_sp_out;

wire NCFSM_d_valid_out, NCFSM_ctrl1_out, NCFSM_wren1_out;
wire NCFSM_SW_enable_650_out, NCFSM_SW_enable_600_out,
NCFSM_start_600_out, NCFSM_start_650_out;
wire[2:0] NCFSM_addr_out;
wire[3:0] NCFSM_ijkd_valid_out;
```

```
wire[1:0] NCFSM_d_ctrl_out;

wire[63:0] Header1_sp_out;
wire[2:0]  Header1_d_out;
wire[63:0] Header2_sp_out;
wire[2:0]  Header2_d_out;
wire[63:0] Header3_sp_out;
wire[2:0]  Header3_d_out;
wire[63:0] Header4_sp_out;
wire[2:0]  Header4_d_out;

wire pd_525, pd_508;
wire[1:0] P_600_kd;
wire[3:0] P_600_jd;
wire[3:0] P_600_j_524_out;
wire[2:0] P_600_d_out;

wire pd_i_full_out;
wire [1:0]  P_650_k_out;
wire[3:0]   P_650_i_out;
wire[2:0]   P_650_d_out;
wire[3:0]   P_650_j_out;
wire [3:0]  P_650_ES_j;
/*-------------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
assign sp_out1 = Mem1_sp_out;
assign sp_out2 = Mem2_sp_out;
assign sp_out3 = Mem3_sp_out;
assign sp_out4 = Mem4_sp_out;
/*-------------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
/*-------------------------------------------------------------*/


/*-----------------Flip Flops------------------------------*/
/*-------------------------------------------------------------*/

/*----------------Module Declarations--------------------*/


/*-------------------------------------------------------------*/


PAC_NCFSM NCFSM(.clock(clock), .reset(reset),
                        .start_600_out(NCFSM_start_600_out),
.start_650_out(NCFSM_start_650_out),
                        .SW_enable_600_out(NCFSM_SW_enable_600_out),
.SW_enable_650_out(NCFSM_SW_enable_650_out),
                        .d_valid_out(NCFSM_d_valid_out),
.d_ctrl_out(NCFSM_d_ctrl_out),
                        .addr_out(NCFSM_addr_out),
                        .ctrl1_out(NCFSM_ctrl1_out),
.ijkd_valid_out(NCFSM_ijkd_valid_out),
                        .wren1_out(NCFSM_wren1_out));
```

```
/*module PAC_NCFSM(clock, reset,
                        start_600_out, start_650_out, SW_enable_600_out,
SW_enable_650_out,
                         d_valid_out, d_ctrl_out,
                        addr_out, ctrl1_out, ijkd_valid_out,
                        wren1_out);        extracted */


// The PAC must have 4 Header pre-processors
PAC_Header Header1(.clock(clock), .reset(reset),
                        .ser_in(ser_in[0]),     /* serial data input*/
                        .d_valid_in(NCFSM_d_valid_out),
                        .sp_out(Header1_sp_out),
                        .d_out(Header1_d_out));
PAC_Header Header2(.clock(clock), .reset(reset),
                        .ser_in(ser_in[1]),     /* serial data input*/
                        .d_valid_in(NCFSM_d_valid_out),
                        .sp_out(Header2_sp_out),
                        .d_out(Header2_d_out));
PAC_Header Header3(.clock(clock), .reset(reset),
                        .ser_in(ser_in[2]),     /* serial data input*/
                        .d_valid_in(NCFSM_d_valid_out),
                        .sp_out(Header3_sp_out),
                        .d_out(Header3_d_out));
PAC_Header Header4(.clock(clock), .reset(reset),
                        .ser_in(ser_in[3]),     /* serial data input*/
                        .d_valid_in(NCFSM_d_valid_out),
                        .sp_out(Header4_sp_out),
                        .d_out(Header4_d_out));

/*PAC_Header(clock, reset,
                        ser_in,
                        d_valid_in,
                        sp_out,
                        d_out);           */

// The PAC has 4 buffers to hold the data as it comes in as 64 bit words
PAC_Memory Mem1(.clock(clock), .reset(reset),   /* ctrl signals*/
                        .sp_in(Header1_sp_out), /* serial-parallel data
input*/
                        .addr_in(NCFSM_addr_out),
.ctrl1_in(NCFSM_ctrl1_out), /* ctrl signals from FSM*/
                        .d_in(P_650_d_out),     .i_in(P_650_i_out),
.j_in(P_650_j_out), .k_in(P_650_k_out),  /* from 650 */
                        .wren1_in(NCFSM_wren1_out),

      .sp_out(Mem1_sp_out),.ijkd_valid_in(NCFSM_ijkd_valid_out[0]) );   /*
PAC_output*/

PAC_Memory Mem2(.clock(clock), .reset(reset),   /* ctrl signals*/
                        .sp_in(Header2_sp_out), /* serial-parallel data
input*/
                        .addr_in(NCFSM_addr_out),
.ctrl1_in(NCFSM_ctrl1_out), /* ctrl signals from FSM*/
                        .d_in(P_650_d_out),     .i_in(P_650_i_out),
.j_in(P_650_j_out), .k_in(P_650_k_out),  /* from 650 */
                        .wren1_in(NCFSM_wren1_out),
```

```
      .sp_out(Mem2_sp_out),.ijkd_valid_in(NCFSM_ijkd_valid_out[1]) );   /*
PAC_output*/

PAC_Memory Mem3(.clock(clock), .reset(reset),   /* ctrl signals*/
                        .sp_in(Header3_sp_out), /* serial-parallel data
input*/
                        .addr_in(NCFSM_addr_out),
.ctrl1_in(NCFSM_ctrl1_out), /* ctrl signals from FSM*/
                        .d_in(P_650_d_out),     .i_in(P_650_i_out),
.j_in(P_650_j_out), .k_in(P_650_k_out),  /* from 650 */
                        .wren1_in(NCFSM_wren1_out),

      .sp_out(Mem3_sp_out),.ijkd_valid_in(NCFSM_ijkd_valid_out[2]) );   /*
PAC_output*/

PAC_Memory Mem4(.clock(clock), .reset(reset),   /* ctrl signals*/
                        .sp_in(Header4_sp_out), /* serial-parallel data
input*/
                        .addr_in(NCFSM_addr_out),
.ctrl1_in(NCFSM_ctrl1_out), /* ctrl signals from FSM*/
                        .d_in(P_650_d_out),     .i_in(P_650_i_out),
.j_in(P_650_j_out), .k_in(P_650_k_out),  /* from 650 */
                        .wren1_in(NCFSM_wren1_out),

      .sp_out(Mem4_sp_out),.ijkd_valid_in(NCFSM_ijkd_valid_out[3]) );   /*
PAC_output*/

/*PAC_Memory(clock, reset,
                        addr_in, wren1_in, ctrl1_in, ijkd_valid_in,
                        sp_in,
                        d_in, j_in, k_in, i_in,
                        sp_out);    */

// First processor in the pipeline
PAC_600 P_600(.clock(clock), .reset(reset),
            .start_600_in(NCFSM_start_600_out),
.SW_enable(NCFSM_SW_enable_600_out),
             .d_ctrl_in(NCFSM_d_ctrl_out),
            .d1_in(Header1_d_out), .d2_in(Header2_d_out),
.d3_in(Header3_d_out), .d4_in(Header4_d_out),
            .ES_j_in(P_650_ES_j), .j_524_out(P_600_j_524_out),
            .jd(P_600_jd), .kd(P_600_kd), .d_out(P_600_d_out));
/* PAC_600(clock, reset,
            start_600_in, SW_enable, d_ctrl_in,
            d1_in, d2_in, d3_in, d4_in,
            ES_j_in,    j_524_out,
            pd_525, pd_508,
            jd, kd, d_out);*/

// Second processor in the pipeline
PAC_650 P_650(.clock(clock), .reset(reset),
            .start_650_in(NCFSM_start_650_out),
.SW_enable(NCFSM_SW_enable_650_out),
            .j_524_in(P_600_j_524_out), .ES_j(P_650_ES_j),
            .j_in(P_600_jd), .k_in(P_600_kd), .d_in(P_600_d_out),
```

```
            .i_out(P_650_i_out),.j_out(P_650_j_out),
.k_out(P_650_k_out),.d_out(P_650_d_out),
            .pd_i_full_out());
/*PAC_650(clock, reset,
            start_650_in, SW_enable,
            j_524_in, ES_j,
            j_in, k_in, d_in,
            i_out, j_out, k_out, d_out,
            pd_i_full_out);         */
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module IIN_H.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Parallel-Serializes the 64 bit data words to 8 bit words.
      This is done to keep the size of the datapath small so the
      switching circuitry is small.

      This module also extracts the packet parameters for
      routing (in the IIN_CB) purposes and for the memory modules.

      This module has a pipeline delay RAM like PAC.  This is done to
      stay consistent with the SW algorithm.

*/
/*-------------------------------------------------------*/
// Uses the IIN_FSM to tell when to capture i,j,k,d & convert the input of
64 bits wide
//    to an output of 8 bits wide
module IIN_H(clock, reset,
                        ijkd_en_in, sp_in, ps_addr_in, ram_addr_in,
wren2_in,
                        ps_out, id_out, jd, kd, dd);        /* outputs */

/*-----------------Port Declarations----------------------*/
input clock, reset, ijkd_en_in, wren2_in;
input[63:0] sp_in;
input[3:0] ps_addr_in;
input[2:0] ram_addr_in;
output[7:0] ps_out;

output[3:0] id_out, jd; // jd, kd, dd used by Memory Modules
output[2:0] dd;
output[1:0] kd;

/*-------------------------------------------------------*/
```

```
/*-----------------Wire & Reg's---------------------------*/
reg [7:0] ps_out;

wire[63:0] ps_conv;
reg[3:0] id_out, jd, id;
reg[2:0] dd;
reg[1:0] kd;
/*--------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/
/*--------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
//  serial-parallelizes the 64 bit data to 8 bits based on ps_addr_in from
the FSM
always @(ps_conv or ps_addr_in)
      casex(ps_addr_in)
            4'd0:begin id_out=id; ps_out=ps_conv[7:0]; end
            4'd1:begin id_out=id; ps_out=ps_conv[15:8]; end
            4'd2:begin id_out=id; ps_out=ps_conv[23:16]; end
            4'd3:begin id_out=id; ps_out=ps_conv[31:24]; end
            4'd4:begin id_out=id; ps_out=ps_conv[39:32]; end
            4'd5:begin id_out=id; ps_out=ps_conv[47:40]; end
            4'd6:begin id_out=id; ps_out=ps_conv[55:48]; end
            4'd7:begin id_out=id; ps_out=ps_conv[63:56]; end
            4'd8:begin id_out=0; ps_out=8'h00; end
            default:begin id_out=0; ps_out=8'hxx; end
      endcase
/*--------------------------------------------------------*/

/*-----------------Flip-Flops-----------------------------*/
always@(posedge clock)
      if(!reset)
            begin
                  dd <= 0;// not needed
                  id <= 0;// only one really needed
                  jd <= 0;// not needed
                  kd <= 0;// not needed
            end
      else
            begin


                  if (ijkd_en_in)
                        begin
                              dd <= ps_conv[2:0];
                              id <= ps_conv[12:9];
                              jd <= ps_conv[8:5];
                              kd <= ps_conv[4:3];
                        end

            end
/*--------------------------------------------------------*/
/*-----------------Module Declarations--------------------*/
ram_pac ram2(           //(8 cycle delay or 1 switch cycle for pipeline)
      .data(sp_in),
      .wraddress(ram_addr_in),
```

```
      .rdaddress(ram_addr_in+1'b1),
      .wren(wren2_in),
      .inclock(clock),
      .outclock(clock),
      .aclr(!reset),
      .q(ps_conv));
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module IIN_FSM.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      This is just a counter that is used to generate addresses
      and control signals for the whole IIN.

*/
/*-------------------------------------------------------*/
module IIN_FSM(clock, reset, id_in1, id_in2, id_in3, id_in4,
                             ps_addr_out,
                             ctrl1_out,
                             ctrl2_out,
                             ctrl3_out,
                             ctrl4_out,
                             ctrl5_out,
                             ctrl6_out,
                             ctrl7_out,
                             ctrl8_out,
                             ijkd_en, wren2_out, ram_addr_out);        /*
outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[3:0] id_in1, id_in2, id_in3, id_in4;
output[3:0] ctrl1_out, ctrl2_out, ctrl3_out, ctrl4_out, ctrl5_out,
ctrl6_out, ctrl7_out, ctrl8_out;
output[3:0] ps_addr_out;
output[2:0] ram_addr_out;
output ijkd_en, wren2_out;
/*-------------------------------------------------------*/

/*-----------------Wire & Reg's--------------------------*/
reg[3:0] ps_addr_out;
reg[8:0] counter;
reg[7:0] c1, c2, c3, c4;
reg ijkd_en;
wire wren2_out;
wire[2:0] ram_addr_out;
/*-------------------------------------------------------*/
```

```
/*-----------------Assigns--------------------------------*/
// Ram2 write signals for pipeline delay
assign wren2_out = (counter[5:0]==6'd0);
assign ram_addr_out = counter[8:6];

// These statements send control signals to each of the memory's Priority
//          Encoders.  This is a simple way to classify how to route
packets.
// Each Mem Module has a 4 input Priority encoder controlled by ctrli_out
where
// i is the Mem Module number
assign ctrl1_out = {    c1[7],
                                        c2[7],
                                        c3[7],
                                        c4[7]};

assign ctrl2_out = {    c1[6],
                                        c2[6],
                                        c3[6],
                                        c4[6]};


assign ctrl3_out = {    c1[5],
                                        c2[5],
                                        c3[5],
                                        c4[5]};


assign ctrl4_out = {    c1[4],
                                        c2[4],
                                        c3[4],
                                        c4[4]};

assign ctrl5_out = {    c1[3],
                                        c2[3],
                                        c3[3],
                                        c4[3]};

assign ctrl6_out = {    c1[2],
                                        c2[2],
                                        c3[2],
                                        c4[2]};

assign ctrl7_out = {    c1[1],
                                        c2[1],
                                        c3[1],
                                        c4[1]};

assign ctrl8_out = {    c1[0],
                                        c2[0],
                                        c3[0],
                                        c4[0]};

/*--------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
```

```
// These statements are control signals to each of the memory modules's
Priority
//          Encoders.  This is a simple way to classify how to route
packets
always@(id_in1)
      casex(id_in1)
            4'd0:c1 = 8'b00000000;
            4'd1:c1 = 8'b10000000;
            4'd2:c1 = 8'b01000000;
            4'd3:c1 = 8'b00100000;
            4'd4:c1 = 8'b00010000;
            4'd5:c1 = 8'b00001000;
            4'd6:c1 = 8'b00000100;
            4'd7:c1 = 8'b00000010;
            4'd8:c1 = 8'b00000001;
            default:c1 = 8'bxxxxxxxx;
      endcase

always@(id_in2)
      casex(id_in2)
            4'd0:c2 = 8'b00000000;
            4'd1:c2 = 8'b10000000;
            4'd2:c2 = 8'b01000000;
            4'd3:c2 = 8'b00100000;
            4'd4:c2 = 8'b00010000;
            4'd5:c2 = 8'b00001000;
            4'd6:c2 = 8'b00000100;
            4'd7:c2 = 8'b00000010;
            4'd8:c2 = 8'b00000001;
            default:c2 = 8'bxxxxxxxx;
      endcase

always@(id_in3)
      casex(id_in3)
            4'd0:c3 = 8'b00000000;
            4'd1:c3 = 8'b10000000;
            4'd2:c3 = 8'b01000000;
            4'd3:c3 = 8'b00100000;
            4'd4:c3 = 8'b00010000;
            4'd5:c3 = 8'b00001000;
            4'd6:c3 = 8'b00000100;
            4'd7:c3 = 8'b00000010;
            4'd8:c3 = 8'b00000001;
            default:c3 = 8'bxxxxxxxx;
      endcase

always@(id_in4)
      casex(id_in4)
            4'd0:c4 = 8'b00000000;
            4'd1:c4 = 8'b10000000;
            4'd2:c4 = 8'b01000000;
            4'd3:c4 = 8'b00100000;
            4'd4:c4 = 8'b00010000;
            4'd5:c4 = 8'b00001000;
            4'd6:c4 = 8'b00000100;
            4'd7:c4 = 8'b00000010;
```

```
            4'd8:c4 = 8'b00000001;
            default:c4 = 8'bxxxxxxxx;
      endcase

/*-------------------------------------------------------*/
always@(counter)
      casex(counter)
            9'b0_0001_0001:begin ijkd_en=1'b1; ps_addr_out=4'd8; end  //
record ijkd parameters in this cycle

            /*9'b0_0001_001x:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
//12h-13h  -> hold each value for 2 cycles
            9'b0_0001_010x:begin ijkd_en=1'b0; ps_addr_out=4'd1; end //
14h-15h
            9'b0_0001_011x:begin ijkd_en=1'b0; ps_addr_out=4'd2; end //
16h-17h
            9'b0_0001_100x:begin ijkd_en=1'b0; ps_addr_out=4'd3; end //
18h-19h
            9'b0_0001_101x:begin ijkd_en=1'b0; ps_addr_out=4'd4; end //
1ah-1bh
            9'b0_0001_110x:begin ijkd_en=1'b0; ps_addr_out=4'd5; end //
1ch-1dh
            9'b0_0001_111x:begin ijkd_en=1'b0; ps_addr_out=4'd6; end //
1eh-1fh
            9'b0_0010_000x:begin ijkd_en=1'b0; ps_addr_out=4'd7; end //
20h-21h*/

            9'bx_0001_010x:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
//(1/0)44h-(1/0)45h also (1/0)c4h-(1/0)c5h
            9'bx_0001_011x:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
//46h-47h
            9'bx_0001_100x:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
//48h-49h
            9'bx_0001_101x:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
//4ah-4bh
            9'bx_0001_110x:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
//4ch-4dh
            9'bx_0001_111x:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
//4eh-4fh
            9'bx_0010_000x:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
//50h-51h
            9'bx_0010_001x:begin ijkd_en=1'b0; ps_addr_out=4'd7; end
//52h-53h

            9'bx_x100_010x:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
//(1/0)44h-(1/0)45h also (1/0)c4h-(1/0)c5h
            9'bx_x100_011x:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
//46h-47h
            9'bx_x100_100x:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
//48h-49h
            9'bx_x100_101x:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
//4ah-4bh
            9'bx_x100_110x:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
//4ch-4dh
            9'bx_x100_111x:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
//4eh-4fh
```

```
            9'bx_x101_000x:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
//50h-51h
            9'bx_x101_001x:begin ijkd_en=1'b0; ps_addr_out=4'd7; end
//52h-53h

            9'bx_1000_010x:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
//(1/0)84h-(1/0)85h
            9'bx_1000_011x:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
//86h-87h
            9'bx_1000_100x:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
//88h-89h
            9'bx_1000_101x:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
//8ah-8bh
            9'bx_1000_110x:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
//8ch-8dh
            9'bx_1000_111x:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
//8eh-8fh
            9'bx_1001_000x:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
//90h-91h
            9'bx_1001_001x:begin ijkd_en=1'b0; ps_addr_out=4'd7; end
//92h-93h

            default begin ijkd_en=1'b0; ps_addr_out=4'd8; end // no packet
signal is ps_addr=8
      endcase

/*
always@(counter)
      casex(counter)
            9'h011:begin ijkd_en=1'b1; ps_addr_out=4'd8; end  // record
ijkd parameters in this cycle
            9'h012:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h013:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h014:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h015:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h016:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h017:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h018:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h019:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h043:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h044:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h045:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h046:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h047:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h048:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h049:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h04a:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h083:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h084:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h085:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h086:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h087:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h088:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h089:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h08a:begin ijkd_en=1'b0; ps_addr_out=4'd7; end
```

```
            9'h0c3:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h0c4:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h0c5:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h0c6:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h0c7:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h0c8:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h0c9:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h0ca:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h103:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h104:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h105:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h106:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h107:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h108:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h109:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h10a:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h143:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h144:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h145:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h146:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h147:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h148:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h149:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h14a:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h183:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h184:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h185:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h186:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h187:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h188:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h189:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h18a:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            9'h1c3:begin ijkd_en=1'b0; ps_addr_out=4'd0; end
            9'h1c4:begin ijkd_en=1'b0; ps_addr_out=4'd1; end
            9'h1c5:begin ijkd_en=1'b0; ps_addr_out=4'd2; end
            9'h1c6:begin ijkd_en=1'b0; ps_addr_out=4'd3; end
            9'h1c7:begin ijkd_en=1'b0; ps_addr_out=4'd4; end
            9'h1c8:begin ijkd_en=1'b0; ps_addr_out=4'd5; end
            9'h1c9:begin ijkd_en=1'b0; ps_addr_out=4'd6; end
            9'h1ca:begin ijkd_en=1'b0; ps_addr_out=4'd7; end

            default begin ijkd_en=1'b0; ps_addr_out=4'd8; end // no packet
signal is ps_addr=8
      endcase
      */
/*-----------------Flip-Flops-----------------------------*/
always@(posedge clock)
      if(!reset)
            begin
                  counter <= 0;
            end
      else
```

```
            begin
                  counter <= counter + 1'b1;
            end
/*-------------------------------------------------------*/

/*-----------------Module Declarations--------------------*/
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module IIN_CB.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      One of theses modules connects to each of the m memory
      modules. It selects which port to connect to the memory module.

      These are connected together to create a 4x8crossbar.
      Each CB is capable of queuing a 2nd packet and interleaving
      it with the 1st packet in the rare case that one module is
      assigned twice.

*/
/*-------------------------------------------------------*/

module IIN_CB(clock, reset, ctrl_in,
                              toggle_dupl_in, // Control from te Memory
Module
                              p1_in, p2_in, p3_in, p4_in,
                              jd1_in, jd2_in, jd3_in, jd4_in,
                              kd1_in, kd2_in, kd3_in, kd4_in,
                              pkt_here, i_dupl_out, line_out,
                              jd_out, kd_out);        /* outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset, toggle_dupl_in;
input[3:0] ctrl_in, jd1_in, jd2_in, jd3_in, jd4_in;
input[1:0] kd1_in, kd2_in, kd3_in, kd4_in;
input[7:0] p1_in, p2_in, p3_in, p4_in;
output pkt_here, i_dupl_out;
output[7:0] line_out;
output[3:0] jd_out;
output[1:0] kd_out;
/*-------------------------------------------------------*/
integer k;
/*-----------------Wire & Reg's--------------------------*/
wire[7:0] line_out;
reg[7:0] line_tmp, line_dupl;
reg pkt_here;
reg[3:0] ctrl_in_f, jd_dupl, jd_tmp;
```

```
wire[3:0] jd_out;
wire[1:0] kd_out;
reg[1:0] kd_dupl, kd_tmp;
wire i_dupl_out;
/*-------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/

// Selection of normal or duplicate packet line
// this is controlled by the Memory Module's toggle_dupl_in signal
assign line_out = (toggle_dupl_in)?line_dupl:line_tmp;
assign jd_out = (toggle_dupl_in)?jd_dupl:jd_tmp;
assign kd_out = (toggle_dupl_in)?kd_dupl:kd_tmp;

// This checks for a duplicate i assignment to this memory module
assign i_dupl_out = (((ctrl_in[3]+ctrl_in[2])+(ctrl_in[1]+ctrl_in[0]))>1);
/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
always@(ctrl_in)
      for(k = 0; k <= 3; k = k + 1)
            ctrl_in_f[k]=ctrl_in[3-k];

always@(ctrl_in or p1_in or p2_in or p3_in or p4_in )
      casex(ctrl_in)
            4'b1xxx:begin pkt_here=1'b1; line_tmp = p1_in; jd_tmp =
jd1_in; kd_tmp = kd1_in; end
            4'b01xx:begin pkt_here=1'b1; line_tmp = p2_in; jd_tmp =
jd2_in; kd_tmp = kd2_in; end
            4'b001x:begin pkt_here=1'b1; line_tmp = p3_in; jd_tmp =
jd3_in; kd_tmp = kd3_in; end
            4'b0001:begin pkt_here=1'b1; line_tmp = p4_in; jd_tmp =
jd4_in; kd_tmp = kd4_in; end
            default:begin pkt_here=1'b0; line_tmp = 8'h00; jd_tmp = 4'h0;
kd_tmp = 4'h0; end
endcase

// The ctrl_in line is reversed here so if there are 2 ctrl_in lines
raised
//    then this will select a different input than the 1st PE.
//          The Memory Module will then select between the 2 if i_dupl_out
is raised
always@(ctrl_in_f or p1_in or p2_in or p3_in or p4_in )
      casex(ctrl_in_f)
            4'b1xxx:begin line_dupl = p4_in; jd_dupl = jd4_in; kd_dupl =
kd4_in; end
            4'b01xx:begin line_dupl = p3_in; jd_dupl = jd3_in; kd_dupl =
kd3_in; end
            4'b001x:begin line_dupl = p2_in; jd_dupl = jd2_in; kd_dupl =
kd2_in; end
            4'b0001:begin line_dupl = p1_in; jd_dupl = jd1_in; kd_dupl =
kd1_in; end
            default:begin line_dupl = 8'h00; jd_dupl = 4'h0; kd_dupl =
4'h0; end
endcase

/*-------------------------------------------------------*/
```

```
/*-----------------Module Declarations---------------------*/
/*---------------------------------------------------------*/
endmodule
```

---

```
/*---------------------------------------------------------*/
/*
Module IIN.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/04

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      This module connects the IIN_CB.v, IIN_H.v & IIN_FSM.v
      in the correct manner to impelent a 4x8 IIN to connect
      between the PAC & the Memory Modules

*/
/*---------------------------------------------------------*/
module IIN(clock, reset,
                        toggle_dupl_in,
                        port1_in,
                        port2_in,
                        port3_in,
                        port4_in,
                        jd_out1, kd_out1, jd_out2, kd_out2,
                        jd_out3, kd_out3, jd_out4, kd_out4,
                        jd_out5, kd_out5, jd_out6, kd_out6,
                        jd_out7, kd_out7, jd_out8, kd_out8,
                        line1_out,
                        line2_out,
                        line3_out,
                        line4_out,
                        line5_out,
                        line6_out,
                        line7_out,
                        line8_out,
                        i_dupl_out,
                        pkt_here_out);
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[7:0] toggle_dupl_in;
input[63:0] port1_in, port2_in, port3_in, port4_in;
output[7:0]       line1_out,
                              line2_out,
                              line3_out,
                              line4_out,
                              line5_out,
                              line6_out,
                              line7_out,
                              line8_out;
```

```
output[7:0] i_dupl_out, pkt_here_out;
output[3:0]  jd_out1, jd_out2, jd_out3, jd_out4,
                        jd_out5, jd_out6, jd_out7, jd_out8;

output[1:0]  kd_out1, kd_out2, kd_out3, kd_out4,
                        kd_out5, kd_out6, kd_out7, kd_out8;
/*-------------------------------------------------------*/

/*-----------------Wire & Reg's--------------------------*/
wire [3:0]  jd_out1, jd_out2, jd_out3, jd_out4,
                        jd_out5, jd_out6, jd_out7, jd_out8;
wire [1:0]  kd_out1, kd_out2, kd_out3, kd_out4,
                        kd_out5, kd_out6, kd_out7, kd_out8;

wire[7:0]   line1_out,
                              line2_out,
                              line3_out,
                              line4_out,
                              line5_out,
                              line6_out,
                              line7_out,
                              line8_out,
                              i_dupl_out,
                              pkt_here_out;

wire[3:0] FSM_ctrl1_out, FSM_ctrl2_out, FSM_ctrl3_out, FSM_ctrl4_out,
FSM_ctrl5_out, FSM_ctrl6_out, FSM_ctrl7_out, FSM_ctrl8_out;
wire[3:0] FSM_ps_addr_out;
wire[2:0] FSM_ram_addr_out;
wire FSM_ijkd_en, FSM_wren2_out;


wire[7:0] H1_ps_out;
wire[3:0] H1_id_out, H1_jd; // jd, kd, dd used by Memory Modules
wire[2:0] H1_dd;
wire[1:0] H1_kd;

wire[7:0] H2_ps_out;
wire[3:0] H2_id_out, H2_jd; // jd, kd, dd used by Memory Modules
wire[2:0] H2_dd;
wire[1:0] H2_kd;

wire[7:0] H3_ps_out;
wire[3:0] H3_id_out, H3_jd; // jd, kd, dd used by Memory Modules
wire[2:0] H3_dd;
wire[1:0] H3_kd;

wire[7:0] H4_ps_out;
wire[3:0] H4_id_out, H4_jd; // jd, kd, dd used by Memory Modules
wire[2:0] H4_dd;
wire[1:0] H4_kd;
/*-------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
/*---------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
```

```
/*-------------------------------------------------------*/

/*-----------------Module Declarations-------------------*/
IIN_FSM FSM(.clock(clock), .reset(reset),
                        .id_in1(H1_id_out), .id_in2(H2_id_out),
.id_in3(H3_id_out), .id_in4(H4_id_out),
                        .ps_addr_out(FSM_ps_addr_out),
                        .ctrl1_out(FSM_ctrl1_out),
                        .ctrl2_out(FSM_ctrl2_out),
                        .ctrl3_out(FSM_ctrl3_out),
                        .ctrl4_out(FSM_ctrl4_out),
                        .ctrl5_out(FSM_ctrl5_out),
                        .ctrl6_out(FSM_ctrl6_out),
                        .ctrl7_out(FSM_ctrl7_out),
                        .ctrl8_out(FSM_ctrl8_out),

      .ijkd_en(FSM_ijkd_en),.ram_addr_out(FSM_ram_addr_out),.wren2_out(FSM
_wren2_out));
/*module
IIN_FSM(clock, reset, id_in1, id_in2, id_in3, id_in4,
                              ps_addr_out,
                              ctrl1_out,
                              ctrl2_out,
                              ctrl3_out,
                              ctrl4_out,
                              ctrl5_out,
                              ctrl6_out,
                              ctrl7_out,
                              ctrl8_out,
                              ijkd_en, wren2_out, ram_addr_out);        */


IIN_H H1(.clock(clock), .reset(reset),
.ijkd_en_in(FSM_ijkd_en),
.sp_in(port1_in), .wren2_in(FSM_wren2_out),
.ram_addr_in(FSM_ram_addr_out),
.ps_addr_in(FSM_ps_addr_out),
.ps_out(H1_ps_out),.id_out(H1_id_out),.jd(H1_jd),.kd(H1_kd),.dd(H1_dd));

IIN_H H2(.clock(clock), .reset(reset),
.ijkd_en_in(FSM_ijkd_en),
.sp_in(port2_in),.wren2_in(FSM_wren2_out), .ram_addr_in(FSM_ram_addr_out),
.ps_addr_in(FSM_ps_addr_out),
.ps_out(H2_ps_out),.id_out(H2_id_out),.jd(H2_jd),.kd(H2_kd),.dd(H2_dd));

IIN_H H3(.clock(clock), .reset(reset),
.ijkd_en_in(FSM_ijkd_en),
.sp_in(port3_in),.wren2_in(FSM_wren2_out), .ram_addr_in(FSM_ram_addr_out),
.ps_addr_in(FSM_ps_addr_out),
.ps_out(H3_ps_out),.id_out(H3_id_out),.jd(H3_jd),.kd(H3_kd),.dd(H3_dd));

IIN_H H4(.clock(clock), .reset(reset),
.ijkd_en_in(FSM_ijkd_en),
.sp_in(port4_in),.wren2_in(FSM_wren2_out), .ram_addr_in(FSM_ram_addr_out),
.ps_addr_in(FSM_ps_addr_out),
.ps_out(H4_ps_out),.id_out(H4_id_out),.jd(H4_jd),.kd(H4_kd),.dd(H4_dd));
```

```
/*

IIN_H(clock, reset,
            ijkd_en_in, sp_in,
            ps_addr_in, ram_addr_in,
            wren2_in,
            ps_out, id_out, jd, kd, dd);
);    */


IIN_CB CB1(.clock(clock), .reset(reset),  /*wire count starts at 0*/
.ctrl_in(FSM_ctrl1_out),
.toggle_dupl_in(toggle_dupl_in[0]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[0]), .i_dupl_out(i_dupl_out[0]),
.line_out(line1_out),
.jd_out(jd_out1), .kd_out(kd_out1));

IIN_CB CB2(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl2_out),
.toggle_dupl_in(toggle_dupl_in[1]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[1]), .i_dupl_out(i_dupl_out[1]),
.line_out(line2_out),
.jd_out(jd_out2), .kd_out(kd_out2));

IIN_CB CB3(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl3_out),
.toggle_dupl_in(toggle_dupl_in[2]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[2]), .i_dupl_out(i_dupl_out[2]),
.line_out(line3_out),
.jd_out(jd_out3), .kd_out(kd_out3));

IIN_CB CB4(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl4_out),
.toggle_dupl_in(toggle_dupl_in[3]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[3]), .i_dupl_out(i_dupl_out[3]),
.line_out(line4_out),
.jd_out(jd_out4), .kd_out(kd_out4));

IIN_CB CB5(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl5_out),
.toggle_dupl_in(toggle_dupl_in[4]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
```

```
.pkt_here(pkt_here_out[4]), .i_dupl_out(i_dupl_out[4]),
.line_out(line5_out),
.jd_out(jd_out5), .kd_out(kd_out5));

IIN_CB CB6(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl6_out),
.toggle_dupl_in(toggle_dupl_in[5]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[5]), .i_dupl_out(i_dupl_out[5]),
.line_out(line6_out),
.jd_out(jd_out6), .kd_out(kd_out6));

IIN_CB CB7(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl7_out),
.toggle_dupl_in(toggle_dupl_in[6]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[6]), .i_dupl_out(i_dupl_out[6]),
.line_out(line7_out),
.jd_out(jd_out7), .kd_out(kd_out7));

IIN_CB CB8(.clock(clock), .reset(reset),
.ctrl_in(FSM_ctrl8_out),
.toggle_dupl_in(toggle_dupl_in[7]),
.p1_in(H1_ps_out),.p2_in(H2_ps_out),.p3_in(H3_ps_out),.p4_in(H4_ps_out),
.jd1_in(H1_jd), .jd2_in(H2_jd), .jd3_in(H3_jd), .jd4_in(H4_jd),
.kd1_in(H1_kd), .kd2_in(H2_kd), .kd3_in(H3_kd), .kd4_in(H4_kd),
.pkt_here(pkt_here_out[7]), .i_dupl_out(i_dupl_out[7]),
.line_out(line8_out),
.jd_out(jd_out8), .kd_out(kd_out8));
/*
IIN_CB(clock, reset, ctrl_in,
                              toggle_dupl_in, // Control from the Memory
Module
                              p1_in, p2_in, p3_in, p4_in,
                              pkt_here,
                              i_dupl_out,
                              line_out);  */
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module MM_RAM.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.
```

```
Module Description:
      Uses SW_Counter (in the upper level module) to update the
      scan table.  Also records the packet in the memory module
      based on the OSA & j,k signals from the upper level module
      that arrive from IIN.

*/
/*-------------------------------------------------------*/
module MM_RAM(clock, reset,
                        line_in,
                        i_dupl_in, pkt_here_in,
                        read_en_in, update_in,
                        jd_in, SW_j_in,
                        kd_in, SW_k_in,
                        toggle_dupl_out,
                        pkt_here_out,
                        f_read_out);            /* outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[7:0] line_in;
input i_dupl_in, pkt_here_in, read_en_in, update_in;
input[3:0] jd_in, SW_j_in;
input[1:0] kd_in, SW_k_in;
output[7:0] f_read_out;
output toggle_dupl_out;
output pkt_here_out;
/*-------------------------------------------------------*/

parameter SIGMAm1=11;
integer k1;
/*-----------------Wire & Reg's--------------------------*/


reg[1:0] OSA [0:(SIGMAm1)];

reg read_en_d;
wire toggle_dupl_out, wren;
reg[6:0] w_addr_reg;
reg[5:0] r_addr_reg;
wire[3:0] jd_addr, SW_j_addr;
wire[9:0] wr_addr, rd_addr;
wire[7:0] read_data_out;
wire[7:0] f_read_out;
wire pkt_here_out;
/*-------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/
assign SW_j_addr = SW_j_in-1'b1;
assign rd_addr = {SW_j_addr, r_addr_reg[5:0]};
// Write function assigns:
assign toggle_dupl_out = (i_dupl_in)?w_addr_reg[0]:0;

assign wren = pkt_here_in;
assign jd_addr= (jd_in>0)?jd_in-1'b1:0;

assign wr_addr = {jd_addr, w_addr_reg[6:1]};
```

```
assign f_read_out = (pkt_here_out)?read_data_out:0;
assign pkt_here_out = ((read_en_d)&&(OSA[SW_j_addr]==SW_k_in));
/*-------------------------------------------------------*/

/*-----------------Combinational logic -------------------*/
/*-------------------------------------------------------*/

/*-----------------Flip Flops ----------------------------*/
always@(posedge clock)
      if(!reset)
            begin
                  for(k1 = 0; k1 <= 11; k1 = k1 + 1)
                        OSA[k1]<=0;
                  r_addr_reg <= 0;
                  w_addr_reg <= 0;
                  read_en_d<=0;
            end
      else if(pkt_here_in)
            begin
                  w_addr_reg <= w_addr_reg+1'b1;  // lower bit is
toggle_dupl_out bit!
                  if((w_addr_reg==0)||(w_addr_reg==1))  // add packet to
OSA
                        OSA[jd_addr]= kd_in;
            end
      else if(read_en_in)
            begin
                  read_en_d<=1;           // supplies a delay to line up
with delay of registered RAM module
                  r_addr_reg <= r_addr_reg+1'b1;
            end
      else if(update_in)
            begin
                  read_en_d<=0;
                  r_addr_reg <= 0;
                  w_addr_reg <= 0;  // since read only runs for 64 cycles
(needs 128 exactly to clear it)
                  if(OSA[SW_j_addr]== SW_k_in)
                        OSA[SW_j_addr] = 0;
            end
      else
            begin
                  read_en_d<=0;  // We do not want this to stay hi or the
data will be bad!
            end

/*-------------------------------------------------------*/

/*-----------------Module Declarations--------------------*/
MEM_MOD MM_ram(line_in,
                              wr_addr,
                              rd_addr,
                              wren,
                              clock,
                              !reset,
                              read_data_out);
```

```
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module MM_FSM.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Another counter that sends out control signals.

*/
/*-------------------------------------------------------*/
module MM_FSM(clock, reset, update, read_en_out);           /* outputs */

/*-----------------Port Declarations----------------------*/
input clock, reset;
output update, read_en_out;
/*-------------------------------------------------------*/

/*-----------------Wire & Reg's---------------------------*/
reg[10:0]counter;
reg delay;
wire read_en_out;
wire update;
wire[8:0]c;
/*-------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/

assign c=counter[8:0];

//assign update = (delay && (counter[8:0] == 9'h1e0));  //x1de is the
earliest
assign update = (delay && (counter[8:0] == 9'h000));  //x1de is the
earliest
/*assign read_en_out = (delay &&
                              (( (c>=9'h024)&&(c<=9'h02b) )
                              ||( (c>=9'h054)&&(c<=9'h05b) )
                              ||( (c>=9'h094)&&(c<=9'h09b) )
                              ||( (c>=9'h0d4)&&(c<=9'h0db) )
                              ||( (c>=9'h124)&&(c<=9'h12b) )
                              ||( (c>=9'h154)&&(c<=9'h15b) )
                              ||( (c>=9'h194)&&(c<=9'h19b) )
                              ||( (c>=9'h1d4)&&(c<=9'h1db) )));*/

assign read_en_out = (delay &&
                              (( (c>=9'h024)&&(c<=9'h02b) )
                              ||( (c>=9'h064)&&(c<=9'h06b) )
```

```
                              ||( (c>=9'h0a4)&&(c<=9'h0ab) )
                              ||( (c>=9'h0e4)&&(c<=9'h0eb) )
                              ||( (c>=9'h124)&&(c<=9'h12b) )
                              ||( (c>=9'h164)&&(c<=9'h16b) )
                              ||( (c>=9'h1a4)&&(c<=9'h1ab) )
                              ||( (c>=9'h1e4)&&(c<=9'h1eb) )));

/*-------------------------------------------------------*/


/*-------------------------------------------------------*/

/*-------------------Flip Flops -----------------------------*/
always@(posedge clock)
      if(!reset)
            begin
                  counter <= 0;
                  delay <= 0;
            end
      else if(counter[10:9] == 2'd2)
            begin
                  delay <= 1;
                  counter <= counter + 1'b1;
            end
      else
            begin
                  counter <= counter + 1'b1;
            end

/*-------------------------------------------------------*/


/*-----------------Module Declarations--------------------*/
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module MM.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
            Connects the memory modules & FSM together.  Also provides
            SWCounter signals to all memory modules tokeep track of the
OSA.
*/
```

```
/*-------------------------------------------------------*/
module MM(clock, reset,
                        line_in1, line_in2, line_in3, line_in4,
                        line_in5, line_in6, line_in7, line_in8,
                        i_dupl_in, pkt_here_in,
                        jd_in1, kd_in1, jd_in2, kd_in2,
                        jd_in3, kd_in3, jd_in4, kd_in4,
                        jd_in5, kd_in5, jd_in6, kd_in6,
                        jd_in7, kd_in7, jd_in8, kd_in8,
                        toggle_dupl_out,
                        pkt_here_out,
                        out1, out2, out3, out4,
                        out5, out6, out7, out8);            /* outputs */

/*-----------------Port Declarations----------------------*/
input clock, reset;

input[7:0]  line_in1, line_in2, line_in3, line_in4,
                        line_in5, line_in6, line_in7, line_in8;

input[3:0]  jd_in1, jd_in2, jd_in3, jd_in4,
                        jd_in5, jd_in6, jd_in7, jd_in8;

input[1:0]  kd_in1, kd_in2, kd_in3, kd_in4,
                        kd_in5, kd_in6, kd_in7, kd_in8;

input [7:0]       i_dupl_in, pkt_here_in;


output[7:0]       toggle_dupl_out, pkt_here_out,
                        out1, out2, out3, out4,
                        out5, out6, out7, out8;

/*-------------------------------------------------------*/

/*-----------------Wire & Reg's---------------------------*/
wire[7:0]   toggle_dupl_out,pkt_here_out,
                        out1, out2, out3, out4,
                        out5, out6, out7, out8;

wire[3:0] SWC_SW_j;
wire [1:0]SWC_SW_k;

wire[7:0] M1_f_read_out;
wire M1_toggle_dupl_out;
wire M1_pkt_here_out;

wire[7:0] M2_f_read_out;
wire M2_toggle_dupl_out;
wire M2_pkt_here_out;

wire[7:0] M3_f_read_out;
wire M3_toggle_dupl_out;
wire M3_pkt_here_out;

wire[7:0] M4_f_read_out;
wire M4_toggle_dupl_out;
```

```
wire M4_pkt_here_out;

wire[7:0] M5_f_read_out;
wire M5_toggle_dupl_out;
wire M5_pkt_here_out;

wire[7:0] M6_f_read_out;
wire M6_toggle_dupl_out;
wire M6_pkt_here_out;

wire[7:0] M7_f_read_out;
wire M7_toggle_dupl_out;
wire M7_pkt_here_out;

wire[7:0] M8_f_read_out;
wire M8_toggle_dupl_out;
wire M8_pkt_here_out;

wire MFSM_update, MFSM_read_en_out;
/*-------------------------------------------------------*/

/*-----------------Assigns---------------------------------*/
/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
/*-------------------------------------------------------*/

/*-----------------Module Declarations--------------------*/

MM_RAM      M1(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in1),
                        .i_dupl_in(i_dupl_in[0]),
.pkt_here_in(pkt_here_in[0]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in1), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in1), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[0]),
                        .pkt_here_out(pkt_here_out[0]),
                        .f_read_out(out1));

MM_RAM      M2(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in2),
                        .i_dupl_in(i_dupl_in[1]),
.pkt_here_in(pkt_here_in[1]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in2), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in2), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[1]),
                        .pkt_here_out(pkt_here_out[1]),
                        .f_read_out(out2));

MM_RAM      M3(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in3),
                        .i_dupl_in(i_dupl_in[2]),
.pkt_here_in(pkt_here_in[2]),
```

```
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in3), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in3), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[2]),
                        .pkt_here_out(pkt_here_out[2]),
                        .f_read_out(out3));

MM_RAM      M4(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in4),
                        .i_dupl_in(i_dupl_in[3]),
.pkt_here_in(pkt_here_in[3]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in4), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in4), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[3]),
                        .pkt_here_out(pkt_here_out[3]),
                        .f_read_out(out4));

MM_RAM      M5(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in5),
                        .i_dupl_in(i_dupl_in[4]),
.pkt_here_in(pkt_here_in[4]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in5), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in5), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[4]),
                        .pkt_here_out(pkt_here_out[4]),
                        .f_read_out(out5));

MM_RAM      M6(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in6),
                        .i_dupl_in(i_dupl_in[5]),
.pkt_here_in(pkt_here_in[5]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in6), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in6), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[5]),
                        .pkt_here_out(pkt_here_out[5]),
                        .f_read_out(out6));

MM_RAM      M7(.clock(clock), .reset(reset),   /*wire count starts at 0*/
                        .line_in(line_in7),
                        .i_dupl_in(i_dupl_in[6]),
.pkt_here_in(pkt_here_in[6]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in7), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in7), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[6]),
                        .pkt_here_out(pkt_here_out[6]),
                        .f_read_out(out7));

MM_RAM      M8(.clock(clock), .reset(reset),   /*wire count starts at 0*/
```

```
                        .line_in(line_in8),
                        .i_dupl_in(i_dupl_in[7]),
.pkt_here_in(pkt_here_in[7]),
                        .read_en_in(MFSM_read_en_out),
.update_in(MFSM_update),
                        .jd_in(jd_in8), .SW_j_in(SWC_SW_j),
                        .kd_in(kd_in8), .SW_k_in(SWC_SW_k),
                        .toggle_dupl_out(toggle_dupl_out[7]),
                        .pkt_here_out(pkt_here_out[7]),
                        .f_read_out(out8));

MM_FSM      MFSM(.clock(clock), .reset(reset), .update(MFSM_update),
.read_en_out(MFSM_read_en_out));

PAC_SWCounter SWC(.clock(clock), .reset(reset), .enable(MFSM_update),
.SW_j(SWC_SW_j), .SW_k(SWC_SW_k));
/*-------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module OIN.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Connects the Memory modules to the parallel-serializer,
      which streams the data out serially.

*/
/*-------------------------------------------------------*/
module OIN(clock, reset, pkt_here_in,
                        line_in1,
                        line_in2,
                        line_in3,
                        line_in4,
                        line_in5,
                        line_in6,
                        line_in7,
                        line_in8,
                        out_port1,
                        out_port2,
                        out_port3,
                        out_port4,
                        pkt_here_out);          /* outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[7:0] pkt_here_in, line_in1, line_in2, line_in3, line_in4,
                                                line_in5, line_in6,
line_in7, line_in8;
```

```
output[7:0] out_port1,
                        out_port2,
                        out_port3,
                        out_port4;

output[3:0] pkt_here_out;
/*-------------------------------------------------------*/

/*-----------------Wire & Reg's--------------------------*/
reg[2:0] dd1, dd2, dd3, dd4, dd5, dd6, dd7, dd8;
wire[2:0] dd1_f, dd2_f, dd3_f, dd4_f, dd5_f, dd6_f, dd7_f, dd8_f;
reg[5:0] counter;
reg[7:0]    out_port1,
                        out_port2,
                        out_port3,
                        out_port4;

wire[7:0]   ctrl1,
                        ctrl2,
                        ctrl3,
                        ctrl4;

wire[3:0] pkt_here_out, pkt_here_temp;

reg[3:0] c1,c2,c3,c4,c5,c6,c7,c8;
/*-------------------------------------------------------*/

/*-----------------Assigns-------------------------------*/

// need a pkt_here_out signal to indicate when there is data
assign pkt_here_temp = ((c1|c2)|(c3|c4))|((c5|c6)|(c7|c8));
// must be reversed to be correct

assign pkt_here_out =
{pkt_here_temp[0],pkt_here_temp[1],pkt_here_temp[2],pkt_here_temp[3]};

// ddn_f assigns avoid FF delay on 1st cycle

assign dd1_f=((counter==0)&&(pkt_here_in[0]))?line_in1[2:0]:dd1;
assign dd2_f=((counter==0)&&(pkt_here_in[1]))?line_in2[2:0]:dd2;
assign dd3_f=((counter==0)&&(pkt_here_in[2]))?line_in3[2:0]:dd3;
assign dd4_f=((counter==0)&&(pkt_here_in[3]))?line_in4[2:0]:dd4;
assign dd5_f=((counter==0)&&(pkt_here_in[4]))?line_in5[2:0]:dd5;
assign dd6_f=((counter==0)&&(pkt_here_in[5]))?line_in6[2:0]:dd6;
assign dd7_f=((counter==0)&&(pkt_here_in[6]))?line_in7[2:0]:dd7;
assign dd8_f=((counter==0)&&(pkt_here_in[7]))?line_in8[2:0]:dd8;

// Priority Encoder routing control signals
assign ctrl1 = {  c1[3],
                                        c2[3],
                                        c3[3],
                                        c4[3],
                                        c5[3],
                                        c6[3],
                                        c7[3],
                                        c8[3]};
```

```
assign ctrl2 = {  c1[2],
                                          c2[2],
                                          c3[2],
                                          c4[2],
                                          c5[2],
                                          c6[2],
                                          c7[2],
                                          c8[2]};

assign ctrl3 = {  c1[1],
                                          c2[1],
                                          c3[1],
                                          c4[1],
                                          c5[1],
                                          c6[1],
                                          c7[1],
                                          c8[1]};

assign ctrl4 = {  c1[0],
                                          c2[0],
                                          c3[0],
                                          c4[0],
                                          c5[0],
                                          c6[0],
                                          c7[0],
                                          c8[0]};
/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
// uses the d on each input to create routing signals
always @(dd1_f)
      case(dd1_f)
            3'd1:       c1 = 8'b1000;
            3'd2:       c1 = 8'b0100;
            3'd3:       c1 = 8'b0010;
            3'd4:       c1 = 8'b0001;
            default:    c1 = 8'b0000;
      endcase

always @(dd2_f)
      case(dd2_f)
            3'd1:       c2 = 8'b1000;
            3'd2:       c2 = 8'b0100;
            3'd3:       c2 = 8'b0010;
            3'd4:       c2 = 8'b0001;
            default:    c2 = 8'b0000;
      endcase

always @(dd3_f)
      case(dd3_f)
            3'd1:       c3 = 8'b1000;
            3'd2:       c3 = 8'b0100;
            3'd3:       c3 = 8'b0010;
            3'd4:       c3 = 8'b0001;
            default:    c3 = 8'b0000;
      endcase
```

```
always @(dd4_f)
      case(dd4_f)
            3'd1:       c4 = 8'b1000;
            3'd2:       c4 = 8'b0100;
            3'd3:       c4 = 8'b0010;
            3'd4:       c4 = 8'b0001;
            default:    c4 = 8'b0000;
      endcase

always @(dd5_f)
      case(dd5_f)
            3'd1:       c5 = 8'b1000;
            3'd2:       c5 = 8'b0100;
            3'd3:       c5 = 8'b0010;
            3'd4:       c5 = 8'b0001;
            default:    c5 = 8'b0000;
      endcase

always @(dd6_f)
      case(dd6_f)
            3'd1:       c6 = 8'b1000;
            3'd2:       c6 = 8'b0100;
            3'd3:       c6 = 8'b0010;
            3'd4:       c6 = 8'b0001;
            default:    c6 = 8'b0000;
      endcase
always @(dd7_f)
      case(dd7_f)
            3'd1:       c7 = 8'b1000;
            3'd2:       c7 = 8'b0100;
            3'd3:       c7 = 8'b0010;
            3'd4:       c7 = 8'b0001;
            default:    c7 = 8'b0000;
      endcase

always @(dd8_f)
      case(dd8_f)
            3'd1:       c8 = 8'b1000;
            3'd2:       c8 = 8'b0100;
            3'd3:       c8 = 8'b0010;
            3'd4:       c8 = 8'b0001;
            default:    c8 = 8'b0000;
      endcase

// routes the packet according to the destination port
always@(ctrl1 or line_in1 or line_in2 or line_in3 or line_in4 or line_in5
or line_in6 or line_in7 or line_in8)
      casex(ctrl1)
            8'b1xxxxxxx: out_port1 = line_in1;
            8'b01xxxxxx: out_port1 = line_in2;
            8'b001xxxxx: out_port1 = line_in3;
            8'b0001xxxx: out_port1 = line_in4;
            8'b00001xxx: out_port1 = line_in5;
            8'b000001xx: out_port1 = line_in6;
            8'b0000001x: out_port1 = line_in7;
            8'b00000001: out_port1 = line_in8;
```

```
            default:out_port1 = 0;
      endcase

always@(ctrl2 or line_in1 or line_in2 or line_in3 or line_in4 or line_in5
or line_in6 or line_in7 or line_in8)
      casex(ctrl2)
            8'b1xxxxxxx: out_port2 = line_in1;
            8'b01xxxxxx: out_port2 = line_in2;
            8'b001xxxxx: out_port2 = line_in3;
            8'b0001xxxx: out_port2 = line_in4;
            8'b00001xxx: out_port2 = line_in5;
            8'b000001xx: out_port2 = line_in6;
            8'b0000001x: out_port2 = line_in7;
            8'b00000001: out_port2 = line_in8;
            default:out_port2 = 0;
      endcase

always@(ctrl3 or line_in1 or line_in2 or line_in3 or line_in4 or line_in5
or line_in6 or line_in7 or line_in8)
      casex(ctrl3)
            8'b1xxxxxxx: out_port3 = line_in1;
            8'b01xxxxxx: out_port3 = line_in2;
            8'b001xxxxx: out_port3 = line_in3;
            8'b0001xxxx: out_port3 = line_in4;
            8'b00001xxx: out_port3 = line_in5;
            8'b000001xx: out_port3 = line_in6;
            8'b0000001x: out_port3 = line_in7;
            8'b00000001: out_port3 = line_in8;
            default:out_port3 = 0;
      endcase

always@(ctrl4 or line_in1 or line_in2 or line_in3 or line_in4 or line_in5
or line_in6 or line_in7 or line_in8)
      casex(ctrl4)
            8'b1xxxxxxx: out_port4 = line_in1;
            8'b01xxxxxx: out_port4 = line_in2;
            8'b001xxxxx: out_port4 = line_in3;
            8'b0001xxxx: out_port4 = line_in4;
            8'b00001xxx: out_port4 = line_in5;
            8'b000001xx: out_port4 = line_in6;
            8'b0000001x: out_port4 = line_in7;
            8'b00000001: out_port4 = line_in8;
            default:out_port4 = 0;
      endcase


/*-------------------------------------------------------*/

/*------------------Flip-Flops---------------------------*/
always@(posedge clock)
      begin
            if(!reset)
                  begin
                        counter <=0;
                        dd1 <= 0;   dd2 <= 0;
                        dd3 <= 0;   dd4 <= 0;
                        dd5 <= 0;   dd6 <= 0;
```

```
                        dd7 <= 0;   dd8 <= 0;
                  end
            else if (pkt_here_in > 0)
                  begin
                        if(counter == 0)
                              begin
                                    dd1 <=
(pkt_here_in[0])?line_in1[2:0]:0;   // extract d & put into dd
                                    dd2 <=
(pkt_here_in[1])?line_in2[2:0]:0;   // extract d & put into dd
                                    dd3 <=
(pkt_here_in[2])?line_in3[2:0]:0;   // extract d & put into dd
                                    dd4 <=
(pkt_here_in[3])?line_in4[2:0]:0;   // extract d & put into dd
                                    dd5 <=
(pkt_here_in[4])?line_in5[2:0]:0;   // extract d & put into dd
                                    dd6 <=
(pkt_here_in[5])?line_in6[2:0]:0;   // extract d & put into dd
                                    dd7 <=
(pkt_here_in[6])?line_in7[2:0]:0;   // extract d & put into dd
                                    dd8 <=
(pkt_here_in[7])?line_in8[2:0]:0;   // extract d & put into dd
                              end
                        counter <= counter + 1'b1;
                  end
            else if(counter==6'd0)
                  begin
                        counter <=0;
                        dd1 <= 0;   dd2 <= 0;
                        dd3 <= 0;   dd4 <= 0;
                        dd5 <= 0;   dd6 <= 0;
                        dd7 <= 0;   dd8 <= 0;
                  end
      end
/*-------------------------------------------------------------*/

/*-----------------Module Declarations---------------------*/
/*-------------------------------------------------------------*/
endmodule
```

---

```
/*-------------------------------------------------------*/
/*
Module PS.v
(c) 2004 B.R. Phelps, North Carolina State University
Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      Seriallizes the 8 bit data to a uniform serail stream.
      This is done so that as few FPGA pins are used as possible

*/
```

```
/*-------------------------------------------------------*/
module PS(clock, reset,
                        pkt_here_in,
                        line1_in,
                        line2_in,
                        line3_in,
                        line4_in,
                        port1_out,
                        port2_out,
                        port3_out,
                        port4_out,
                              pkt_here_out);          /* outputs */
/*-----------------Port Declarations----------------------*/
input clock, reset;
input[3:0] pkt_here_in;

input[7:0]  line1_in,
                        line2_in,
                        line3_in,
                        line4_in;

output[3:0] pkt_here_out;

output            port1_out,
                        port2_out,
                        port3_out,
                        port4_out;
/*-------------------------------------------------------*/
integer k;

/*-----------------Wire & Reg's---------------------------*/
reg         port1_out,
                  port2_out,
                  port3_out,
                  port4_out;

wire[3:0] pkt_here_out;

reg[5:0] counter;

reg p1_here, p2_here, p3_here, p4_here;

wire[7:0]   ps1f,
                        ps2f,
                        ps3f,
                        ps4f;

reg[7:0]    ps1[0:7];
reg[7:0]    ps2[0:7];
reg[7:0]    ps3[0:7];
reg[7:0]    ps4[0:7];
/*-------------------------------------------------------*/

/*-----------------Assigns--------------------------------*/
// remove delay of output with 2:1 mux
assign ps1f = (counter==0)?line1_in:ps1[counter[5:3]];
assign ps2f = (counter==0)?line2_in:ps2[counter[5:3]];
```

```
assign ps3f = (counter==0)?line3_in:ps3[counter[5:3]];
assign ps4f = (counter==0)?line4_in:ps4[counter[5:3]];

assign pkt_here_out = (counter==0)?{pkt_here_in[3], pkt_here_in[2],
pkt_here_in[1], pkt_here_in[0]}:{p4_here, p3_here, p2_here,
p1_here};//{p4_here|pkt_here_in[3], p3_here|pkt_here_in[2],
p2_here|pkt_here_in[1], p1_here|pkt_here_in[0]};

/*-------------------------------------------------------*/

/*-----------------Combinational logic --------------------*/
always@(counter[2:0] or ps1f or ps2f or ps3f or ps4f)
      case(counter[2:0])
            3'd0:begin port1_out = ps1f[0]; port2_out = ps2f[0]; port3_out
= ps3f[0]; port4_out = ps4f[0]; end
            3'd1:begin port1_out = ps1f[1]; port2_out = ps2f[1]; port3_out
= ps3f[1]; port4_out = ps4f[1]; end
            3'd2:begin port1_out = ps1f[2]; port2_out = ps2f[2]; port3_out
= ps3f[2]; port4_out = ps4f[2]; end
            3'd3:begin port1_out = ps1f[3]; port2_out = ps2f[3]; port3_out
= ps3f[3]; port4_out = ps4f[3]; end
            3'd4:begin port1_out = ps1f[4]; port2_out = ps2f[4]; port3_out
= ps3f[4]; port4_out = ps4f[4]; end
            3'd5:begin port1_out = ps1f[5]; port2_out = ps2f[5]; port3_out
= ps3f[5]; port4_out = ps4f[5]; end
            3'd6:begin port1_out = ps1f[6]; port2_out = ps2f[6]; port3_out
= ps3f[6]; port4_out = ps4f[6]; end
            3'd7:begin port1_out = ps1f[7]; port2_out = ps2f[7]; port3_out
= ps3f[7]; port4_out = ps4f[7]; end
      endcase
/*-------------------------------------------------------*/

always@(posedge clock)
      if(!reset)
            begin
                  p1_here <= 0; p2_here <= 0; p3_here <= 0; p4_here <= 0;
                  counter<=0;
                  for(k=0; k<=7; k=k+1)
                        begin
                              ps1[k] <= 0; ps2[k] <= 0; ps3[k] <= 0;
ps4[k] <= 0;
                        end
            end
      else if((pkt_here_in > 0)||(counter > 0))
            begin
                  counter <= counter +1'b1;
                  if(pkt_here_in[0]&&(counter<8))
                        begin
                              ps1[counter[2:0]] <= line1_in; p1_here <= 1;
                        end
                  else if (counter==6'd0)
                        begin
                              p1_here <= 0;
                              for(k=0; k<=7; k=k+1)
                                    begin
                                          ps1[k] <= 0;
                                    end
```

```
                        end

                  if(pkt_here_in[1]&&(counter<8))
                        begin
                              ps2[counter[2:0]] <= line2_in; p2_here <= 1;
                        end
                  else if (counter==6'd0)
                        begin
                              p2_here <= 0;
                              for(k=0; k<=7; k=k+1)
                                    begin
                                          ps2[k] <= 0;
                                    end
                        end

                  if(pkt_here_in[2]&&(counter<8))
                        begin
                              ps3[counter[2:0]] <= line3_in; p3_here <= 1;
                        end
                  else if (counter==6'd0)
                        begin
                              p3_here <= 0;
                              for(k=0; k<=7; k=k+1)
                                    begin
                                          ps3[k] <= 0;
                                    end
                        end

                  if(pkt_here_in[3]&&(counter<8))
                        begin
                              ps4[counter[2:0]] <= line4_in; p4_here <= 1;
                        end
                  else if (counter==6'd0)
                        begin
                              p4_here <= 0;
                              for(k=0; k<=7; k=k+1)
                                    begin
                                          ps4[k] <= 0;
                                    end
                        end


            end
      else
            begin
                  for(k=0; k<=7; k=k+1)
                        begin
                              ps1[k] <= 0; ps2[k] <= 0; ps3[k] <= 0;
ps4[k] <= 0;
                        end
            end

/*-----------------Module Declarations--------------------*/
/*--------------------------------------------------------*/
endmodule
```

```
/*-----------------------------------------------------------*/
/*

Module SW_Top_Level.v

Created by: Brian R. Phelps   12/22/03

(c) 2004 B.R. Phelps, North Carolina State University

Project: Sliding Window Packet Switch FPGA Implementation
                 from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
      This uses the SW_Counter module & tables such as LC_j,
      LC_k, and Qd to assign the j and k values according to the d
      value given.

      One aspect of the algorithm for Processor 600 (ES_j_in)
      requires information from the Scan Table which is in PAC_650.
      Since PAC_650 is one cycle behind this module, PAC_650 uses
      predictive logic based on the current j(unassigned) logic
      value from 600 to assign ES_j.

*/
/*-----------------------------------------------------------*/
module SW_Top_Level(clock, reset, ser_in,
                        PS1_port1_out,
                        PS1_port2_out,
                        PS1_port3_out,
                        PS1_port4_out,
                        PS1_pkt_here_out);

input clock, reset;

input[3:0] ser_in;                        /* serial data input*/

output[3:0] PS1_pkt_here_out;

output      PS1_port1_out,
                  PS1_port2_out,
                  PS1_port3_out,
                  PS1_port4_out;


wire d_valid, ctrl1, wren1, wren2, start_600;

wire[63:0] sp;    /* serial-parallel out */
wire[2:0] d, addr;            /* d extracted */
wire[1:0] d_ctrl;

// Declare wires for the top level design
wire[63:0] sp_out1, sp_out2, sp_out3, sp_out4;

wire[7:0]   IIN1_line1_out,
                              IIN1_line2_out,
                              IIN1_line3_out,
                              IIN1_line4_out,
```

```
                              IIN1_line5_out,
                              IIN1_line6_out,
                              IIN1_line7_out,
                              IIN1_line8_out;
wire[7:0]   IIN1_i_dupl_out, IIN1_pkt_here_out;
wire[3:0]  IIN1_jd_out1, IIN1_jd_out2, IIN1_jd_out3, IIN1_jd_out4,
                        IIN1_jd_out5, IIN1_jd_out6, IIN1_jd_out7,
IIN1_jd_out8;

wire[1:0]  IIN1_kd_out1, IIN1_kd_out2, IIN1_kd_out3, IIN1_kd_out4,
                        IIN1_kd_out5, IIN1_kd_out6, IIN1_kd_out7,
IIN1_kd_out8;

wire[7:0]   MM1_toggle_dupl_out, MM1_pkt_here_out,
                        MM1_out1, MM1_out2, MM1_out3, MM1_out4,
                        MM1_out5, MM1_out6, MM1_out7, MM1_out8;

wire[7:0]   OIN1_out_port1,
                        OIN1_out_port2,
                        OIN1_out_port3,
                        OIN1_out_port4;

wire[3:0] OIN1_pkt_here_out;

wire[3:0] PS1_pkt_here_out;

wire        PS1_port1_out,
                  PS1_port2_out,
                  PS1_port3_out,
                  PS1_port4_out;

/*------------------------- Top level module connections    --------------
----------------*/
/*
                  inputs:
                        clock, reset,
                        ser_in[3:0]             // serial packet input


                  outputs:
                        PS1_port1_out           // serial packet output
                        PS1_port2_out
                        PS1_port3_out
                        PS1_port4_out

                        PS1_pkt_here_out[3:0]   // packet present(no
delay)
*/


PAC_Full PAC(.clock(clock), .reset(reset), .ser_in(ser_in),
      .sp_out1(sp_out1), .sp_out2(sp_out2), .sp_out3(sp_out3),
.sp_out4(sp_out4));

IIN IIN1(.clock(clock), .reset(reset),
                        .toggle_dupl_in(MM1_toggle_dupl_out),
                        .port1_in(sp_out1),
```

```
                        .port2_in(sp_out2),
                        .port3_in(sp_out3),
                        .port4_in(sp_out4),
                        .jd_out1(IIN1_jd_out1), .kd_out1(IIN1_kd_out1),
                        .jd_out2(IIN1_jd_out2), .kd_out2(IIN1_kd_out2),
                        .jd_out3(IIN1_jd_out3), .kd_out3(IIN1_kd_out3),
                        .jd_out4(IIN1_jd_out4), .kd_out4(IIN1_kd_out4),
                        .jd_out5(IIN1_jd_out5), .kd_out5(IIN1_kd_out5),
                        .jd_out6(IIN1_jd_out6), .kd_out6(IIN1_kd_out6),
                        .jd_out7(IIN1_jd_out7), .kd_out7(IIN1_kd_out7),
                        .jd_out8(IIN1_jd_out8), .kd_out8(IIN1_kd_out8),
                        .line1_out(IIN1_line1_out),
                        .line2_out(IIN1_line2_out),
                        .line3_out(IIN1_line3_out),
                        .line4_out(IIN1_line4_out),
                        .line5_out(IIN1_line5_out),
                        .line6_out(IIN1_line6_out),
                        .line7_out(IIN1_line7_out),
                        .line8_out(IIN1_line8_out),
                        .i_dupl_out(IIN1_i_dupl_out),
                        .pkt_here_out(IIN1_pkt_here_out));

/*module IIN(clock, reset,
                        toggle_dupl_in,
                        port1_in,
                        port2_in,
                        port3_in,
                        port4_in,
                        jd_out1, kd_out1, jd_out2, kd_out2,
                        jd_out3, kd_out3, jd_out4, kd_out4,
                        jd_out5, kd_out5, jd_out6, kd_out6,
                        jd_out7, kd_out7, jd_out8, kd_out8,
                        line1_out,
                        line2_out,
                        line3_out,
                        line4_out,
                        line5_out,
                        line6_out,
                        line7_out,
                        line8_out,
                        i_dupl_out,
                        pkt_here_out);          */


MM  MM1(.clock(clock),.reset(reset),
                        .line_in1(IIN1_line1_out),
                        .line_in2(IIN1_line2_out),
                        .line_in3(IIN1_line3_out),
                        .line_in4(IIN1_line4_out),
                        .line_in5(IIN1_line5_out),
                        .line_in6(IIN1_line6_out),
                        .line_in7(IIN1_line7_out),
                        .line_in8(IIN1_line8_out),
                        .i_dupl_in(IIN1_i_dupl_out),
.pkt_here_in(IIN1_pkt_here_out),
                        .jd_in1(IIN1_jd_out1), .kd_in1(IIN1_kd_out1),
                        .jd_in2(IIN1_jd_out2), .kd_in2(IIN1_kd_out2),
```

```
                        .jd_in3(IIN1_jd_out3), .kd_in3(IIN1_kd_out3),
                        .jd_in4(IIN1_jd_out4), .kd_in4(IIN1_kd_out4),
                        .jd_in5(IIN1_jd_out5), .kd_in5(IIN1_kd_out5),
                        .jd_in6(IIN1_jd_out6), .kd_in6(IIN1_kd_out6),
                        .jd_in7(IIN1_jd_out7), .kd_in7(IIN1_kd_out7),
                        .jd_in8(IIN1_jd_out8), .kd_in8(IIN1_kd_out8),
                        .toggle_dupl_out(MM1_toggle_dupl_out),
                        .pkt_here_out(MM1_pkt_here_out),
                        .out1(MM1_out1),
                     .out2(MM1_out2),
                        .out3(MM1_out3),
                        .out4(MM1_out4),
                        .out5(MM1_out5),
                        .out6(MM1_out6),
                        .out7(MM1_out7),
                        .out8(MM1_out8));       /* outputs */

OIN   OIN1(.clock(clock),.reset(reset),
                        .pkt_here_in(MM1_pkt_here_out),
                        .line_in1(MM1_out1),
                        .line_in2(MM1_out2),
                        .line_in3(MM1_out3),
                        .line_in4(MM1_out4),
                        .line_in5(MM1_out5),
                        .line_in6(MM1_out6),
                        .line_in7(MM1_out7),
                        .line_in8(MM1_out8),
                        .out_port1(OIN1_out_port1),
                        .out_port2(OIN1_out_port2),
                        .out_port3(OIN1_out_port3),
                        .out_port4(OIN1_out_port4),
                        .pkt_here_out(OIN1_pkt_here_out));        /*
outputs */

PS          PS1(.clock(clock),.reset(reset),
                        .pkt_here_in(OIN1_pkt_here_out),
                        .line1_in(OIN1_out_port1),
                        .line2_in(OIN1_out_port2),
                        .line3_in(OIN1_out_port3),
                        .line4_in(OIN1_out_port4),
                        .port1_out(PS1_port1_out),
                        .port2_out(PS1_port2_out),
                        .port3_out(PS1_port3_out),
                        .port4_out(PS1_port4_out),
                              .pkt_here_out(PS1_pkt_here_out));         /*
outputs */

/*-------------------------------------------------------------------------
------------*/

endmodule
```

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altdpram

// ============================================================
// File Name: ram_pac.v
// Megafunction Name(s):
//                altdpram
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ************************************************************


//Copyright (C) 1991-2003 Altera Corporation
//Any  megafunction  design,  and related netlist (encrypted  or
decrypted),
//support information,  device programming or simulation file,  and any
other
//associated  documentation or information  provided by  Altera  or a
partner
//under  Altera's   Megafunction   Partnership   Program  may  be  used
only
//to program  PLD  devices (but not masked  PLD  devices) from  Altera.
Any
//other  use  of such  megafunction  design,  netlist,  support
information,
//device programming or simulation file,  or any other  related
documentation
//or information  is prohibited  for  any  other purpose,  including, but
not
//limited to  modification,  reverse engineering,  de-compiling, or use
with
//any other  silicon devices,  unless such use is  explicitly  licensed
under
//a separate agreement with  Altera  or a megafunction partner.  Title to
the
//intellectual property,  including patents,  copyrights,  trademarks,
trade
//secrets,  or maskworks,  embodied in any such megafunction design,
netlist,
//support  information,  device programming or simulation file,  or any
other
//related documentation or information provided by  Altera  or a
megafunction
//partner, remains with Altera, the megafunction partner, or their
respective
//licensors. No other licenses, including any licenses needed under any
third
//party's intellectual property, are provided herein.


module ram_pac (
      data,
```

```
	wraddress,
	rdaddress,
	wren,
	inclock,
	outclock,
	aclr,
	q);

	input [63:0]  data;
	input [2:0]  wraddress;
	input [2:0]  rdaddress;
	input   wren;
	input   inclock;
	input   outclock;
	input   aclr;
	output      [63:0]  q;

	wire [63:0] sub_wire0;
	wire [63:0] q = sub_wire0[63:0];

	altdpram    altdpram_component (
				.outclock (outclock),
				.wren (wren),
				.inclock (inclock),
				.aclr (aclr),
				.data (data),
				.rdaddress (rdaddress),
				.wraddress (wraddress),
				.q (sub_wire0));
	defparam
		altdpram_component.intended_device_family = "APEX20KE",
		altdpram_component.width = 64,
		altdpram_component.widthad = 3,
		altdpram_component.indata_reg = "INCLOCK",
		altdpram_component.wraddress_reg = "INCLOCK",
		altdpram_component.wrcontrol_reg = "INCLOCK",
		altdpram_component.rdaddress_reg = "INCLOCK",
		altdpram_component.rdcontrol_reg = "UNREGISTERED",
		altdpram_component.outdata_reg = "OUTCLOCK",
		altdpram_component.indata_aclr = "ON",
		altdpram_component.wraddress_aclr = "ON",
		altdpram_component.wrcontrol_aclr = "ON",
		altdpram_component.rdaddress_aclr = "ON",
		altdpram_component.rdcontrol_aclr = "OFF",
		altdpram_component.outdata_aclr = "ON",
		altdpram_component.lpm_type = "altdpram",
		altdpram_component.use_eab = "ON";


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "1"
// Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
// Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
```

```
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "APEX20KE"
// Retrieval info: PRIVATE: VarWidth NUMERIC "0"
// Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "64"
// Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "64"
// Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "64"
// Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "64"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "512"
// Retrieval info: PRIVATE: Clock NUMERIC "2"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "1"
// Retrieval info: PRIVATE: Clock_A NUMERIC "0"
// Retrieval info: PRIVATE: Clock_B NUMERIC "0"
// Retrieval info: PRIVATE: REGdata NUMERIC "1"
// Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
// Retrieval info: PRIVATE: REGwren NUMERIC "1"
// Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: REGrren NUMERIC "1"
// Retrieval info: PRIVATE: REGq NUMERIC "1"
// Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "1"
// Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "1"
// Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "1"
// Retrieval info: PRIVATE: CLRwren NUMERIC "1"
// Retrieval info: PRIVATE: CLRwraddress NUMERIC "1"
// Retrieval info: PRIVATE: CLRrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: CLRrren NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "1"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: enable NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "0"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: UseLCs NUMERIC "0"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "APEX20KE"
// Retrieval info: CONSTANT: WIDTH NUMERIC "64"
// Retrieval info: CONSTANT: WIDTHAD NUMERIC "3"
// Retrieval info: CONSTANT: INDATA_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: WRADDRESS_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: WRCONTROL_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: RDADDRESS_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: RDCONTROL_REG STRING "UNREGISTERED"
// Retrieval info: CONSTANT: OUTDATA_REG STRING "OUTCLOCK"
// Retrieval info: CONSTANT: INDATA_ACLR STRING "ON"
// Retrieval info: CONSTANT: WRADDRESS_ACLR STRING "ON"
// Retrieval info: CONSTANT: WRCONTROL_ACLR STRING "ON"
// Retrieval info: CONSTANT: RDADDRESS_ACLR STRING "ON"
// Retrieval info: CONSTANT: RDCONTROL_ACLR STRING "OFF"
```

```
// Retrieval info: CONSTANT: OUTDATA_ACLR STRING "ON"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altdpram"
// Retrieval info: CONSTANT: USE_EAB STRING "ON"
// Retrieval info: USED_PORT: data 0 0 64 0 INPUT NODEFVAL data[63..0]
// Retrieval info: USED_PORT: q 0 0 64 0 OUTPUT NODEFVAL q[63..0]
// Retrieval info: USED_PORT: wraddress 0 0 3 0 INPUT NODEFVAL
wraddress[2..0]
// Retrieval info: USED_PORT: rdaddress 0 0 3 0 INPUT NODEFVAL
rdaddress[2..0]
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT VCC wren
// Retrieval info: USED_PORT: inclock 0 0 0 0 INPUT NODEFVAL inclock
// Retrieval info: USED_PORT: outclock 0 0 0 0 INPUT NODEFVAL outclock
// Retrieval info: USED_PORT: aclr 0 0 0 0 INPUT GND aclr
// Retrieval info: CONNECT: @data 0 0 64 0 data 0 0 64 0
// Retrieval info: CONNECT: q 0 0 64 0 @q 0 0 64 0
// Retrieval info: CONNECT: @wraddress 0 0 3 0 wraddress 0 0 3 0
// Retrieval info: CONNECT: @rdaddress 0 0 3 0 rdaddress 0 0 3 0
// Retrieval info: CONNECT: @wren 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: @inclock 0 0 0 0 inclock 0 0 0 0
// Retrieval info: CONNECT: @outclock 0 0 0 0 outclock 0 0 0 0
// Retrieval info: CONNECT: @aclr 0 0 0 0 aclr 0 0 0 0
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
```

---

```
// megafunction wizard: %LPM_SHIFTREG%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_shiftreg

// ============================================================
// File Name: shift_reg2.v
// Megafunction Name(s):
//                lpm_shiftreg
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ************************************************************


//Copyright (C) 1991-2003 Altera Corporation
//Any  megafunction  design,  and related netlist (encrypted  or
decrypted),
//support information,  device programming or simulation file,  and any
other
//associated  documentation or information  provided by  Altera  or a
partner
//under  Altera's   Megafunction   Partnership   Program  may  be  used
only
//to program  PLD  devices (but not masked  PLD  devices) from  Altera.
Any
//other  use  of such  megafunction  design,  netlist,  support
information,
//device programming or simulation file,  or any other  related
documentation
//or information  is prohibited  for  any  other purpose,  including, but
not
```

```
//limited to  modification,  reverse engineering,  de-compiling, or use
with
//any other  silicon devices,  unless such use is  explicitly  licensed
under
//a separate agreement with  Altera  or a megafunction partner.  Title to
the
//intellectual property,  including patents,  copyrights,  trademarks,
trade
//secrets,  or maskworks,  embodied in any such megafunction design,
netlist,
//support  information,  device programming or simulation file,  or any
other
//related documentation or information provided by  Altera  or a
megafunction
//partner, remains with Altera, the megafunction partner, or their
respective
//licensors. No other licenses, including any licenses needed under any
third
//party's intellectual property, are provided herein.


module shift_reg2 (
      clock,
      shiftin,
      sclr,
      q);

      input   clock;
      input   shiftin;
      input   sclr;
      output      [63:0]  q;

      wire [63:0] sub_wire0;
      wire [63:0] q = sub_wire0[63:0];

      lpm_shiftreg      lpm_shiftreg_component (
                        .sclr (sclr),
                        .clock (clock),
                        .shiftin (shiftin),
                        .q (sub_wire0));
      defparam
            lpm_shiftreg_component.lpm_type = "LPM_SHIFTREG",
            lpm_shiftreg_component.lpm_width = 64,
            lpm_shiftreg_component.lpm_direction = "RIGHT";


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: nBit NUMERIC "64"
// Retrieval info: PRIVATE: LeftShift NUMERIC "0"
// Retrieval info: PRIVATE: Q_OUT NUMERIC "1"
// Retrieval info: PRIVATE: SerialShiftOutput NUMERIC "0"
// Retrieval info: PRIVATE: CLK_EN NUMERIC "0"
// Retrieval info: PRIVATE: SerialShiftInput NUMERIC "1"
```

```
// Retrieval info: PRIVATE: ParallelDataInput NUMERIC "0"
// Retrieval info: PRIVATE: SCLR NUMERIC "1"
// Retrieval info: PRIVATE: SLOAD NUMERIC "0"
// Retrieval info: PRIVATE: SSET NUMERIC "0"
// Retrieval info: PRIVATE: SSET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: SSETV NUMERIC "0"
// Retrieval info: PRIVATE: ACLR NUMERIC "0"
// Retrieval info: PRIVATE: ALOAD NUMERIC "0"
// Retrieval info: PRIVATE: ASET NUMERIC "0"
// Retrieval info: PRIVATE: ASET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: ASETV NUMERIC "0"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_SHIFTREG"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "64"
// Retrieval info: CONSTANT: LPM_DIRECTION STRING "RIGHT"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL clock
// Retrieval info: USED_PORT: q 0 0 64 0 OUTPUT NODEFVAL q[63..0]
// Retrieval info: USED_PORT: shiftin 0 0 0 0 INPUT NODEFVAL shiftin
// Retrieval info: USED_PORT: sclr 0 0 0 0 INPUT NODEFVAL sclr
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 64 0 @q 0 0 64 0
// Retrieval info: CONNECT: @shiftin 0 0 0 0 shiftin 0 0 0 0
// Retrieval info: CONNECT: @sclr 0 0 0 0 sclr 0 0 0 0
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
```

---

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altdpram

// ============================================================
// File Name: MEM_MOD.v
// Megafunction Name(s):
//                altdpram
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ************************************************************


//Copyright (C) 1991-2003 Altera Corporation
//Any  megafunction  design,  and related netlist (encrypted  or
decrypted),
//support information,  device programming or simulation file,  and any
other
//associated  documentation or information  provided by  Altera  or a
partner
//under  Altera's   Megafunction   Partnership   Program  may  be  used
only
//to program  PLD  devices (but not masked  PLD  devices) from  Altera.
Any
//other  use  of such  megafunction  design,  netlist,  support
information,
//device programming or simulation file,  or any other  related
documentation
```

```
//or information  is prohibited  for  any  other purpose,  including, but
not
//limited to  modification,  reverse engineering,  de-compiling, or use
with
//any other  silicon devices,  unless such use is  explicitly  licensed
under
//a separate agreement with  Altera  or a megafunction partner.  Title to
the
//intellectual property,  including patents,  copyrights,  trademarks,
trade
//secrets,  or maskworks,  embodied in any such megafunction design,
netlist,
//support  information,  device programming or simulation file,  or any
other
//related documentation or information provided by  Altera  or a
megafunction
//partner, remains with Altera, the megafunction partner, or their
respective
//licensors. No other licenses, including any licenses needed under any
third
//party's intellectual property, are provided herein.


module MEM_MOD (
      data,
      wraddress,
      rdaddress,
      wren,
      clock,
      aclr,
      q);

      input [7:0]  data;
      input [9:0]  wraddress;
      input [9:0]  rdaddress;
      input   wren;
      input   clock;
      input   aclr;
      output      [7:0]  q;

      wire [7:0] sub_wire0;
      wire [7:0] q = sub_wire0[7:0];

      altdpram    altdpram_component (
                        .wren (wren),
                        .inclock (clock),
                        .aclr (aclr),
                        .data (data),
                        .rdaddress (rdaddress),
                        .wraddress (wraddress),
                        .q (sub_wire0));
      defparam
            altdpram_component.intended_device_family = "APEX20KE",
            altdpram_component.width = 8,
            altdpram_component.widthad = 10,
            altdpram_component.indata_reg = "INCLOCK",
            altdpram_component.wraddress_reg = "INCLOCK",
```

```
            altdpram_component.wrcontrol_reg = "INCLOCK",
            altdpram_component.rdaddress_reg = "INCLOCK",
            altdpram_component.rdcontrol_reg = "UNREGISTERED",
            altdpram_component.outdata_reg = "UNREGISTERED",
            altdpram_component.indata_aclr = "ON",
            altdpram_component.wraddress_aclr = "ON",
            altdpram_component.wrcontrol_aclr = "ON",
            altdpram_component.rdaddress_aclr = "ON",
            altdpram_component.rdcontrol_aclr = "OFF",
            altdpram_component.outdata_aclr = "OFF",
            altdpram_component.lpm_type = "altdpram",
            altdpram_component.use_eab = "ON";


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
// Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
// Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "APEX20KE"
// Retrieval info: PRIVATE: VarWidth NUMERIC "0"
// Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "8"
// Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "8"
// Retrieval info: PRIVATE: MEMSIZE NUMERIC "6144"
// Retrieval info: PRIVATE: Clock NUMERIC "0"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "1"
// Retrieval info: PRIVATE: Clock_A NUMERIC "0"
// Retrieval info: PRIVATE: Clock_B NUMERIC "0"
// Retrieval info: PRIVATE: REGdata NUMERIC "1"
// Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
// Retrieval info: PRIVATE: REGwren NUMERIC "1"
// Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: REGrren NUMERIC "1"
// Retrieval info: PRIVATE: REGq NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "1"
// Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "1"
// Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "1"
// Retrieval info: PRIVATE: CLRwren NUMERIC "1"
// Retrieval info: PRIVATE: CLRwraddress NUMERIC "1"
// Retrieval info: PRIVATE: CLRrdaddress NUMERIC "1"
// Retrieval info: PRIVATE: CLRrren NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "1"
// Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
```

```
// Retrieval info: PRIVATE: enable NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "0"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: UseLCs NUMERIC "0"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "APEX20KE"
// Retrieval info: CONSTANT: WIDTH NUMERIC "8"
// Retrieval info: CONSTANT: WIDTHAD NUMERIC "10"
// Retrieval info: CONSTANT: INDATA_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: WRADDRESS_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: WRCONTROL_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: RDADDRESS_REG STRING "INCLOCK"
// Retrieval info: CONSTANT: RDCONTROL_REG STRING "UNREGISTERED"
// Retrieval info: CONSTANT: OUTDATA_REG STRING "UNREGISTERED"
// Retrieval info: CONSTANT: INDATA_ACLR STRING "ON"
// Retrieval info: CONSTANT: WRADDRESS_ACLR STRING "ON"
// Retrieval info: CONSTANT: WRCONTROL_ACLR STRING "ON"
// Retrieval info: CONSTANT: RDADDRESS_ACLR STRING "ON"
// Retrieval info: CONSTANT: RDCONTROL_ACLR STRING "OFF"
// Retrieval info: CONSTANT: OUTDATA_ACLR STRING "OFF"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altdpram"
// Retrieval info: CONSTANT: USE_EAB STRING "ON"
// Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL data[7..0]
// Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL q[7..0]
// Retrieval info: USED_PORT: wraddress 0 0 10 0 INPUT NODEFVAL
wraddress[9..0]
// Retrieval info: USED_PORT: rdaddress 0 0 10 0 INPUT NODEFVAL
rdaddress[9..0]
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT VCC wren
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL clock
// Retrieval info: USED_PORT: aclr 0 0 0 0 INPUT GND aclr
// Retrieval info: CONNECT: @data 0 0 8 0 data 0 0 8 0
// Retrieval info: CONNECT: q 0 0 8 0 @q 0 0 8 0
// Retrieval info: CONNECT: @wraddress 0 0 10 0 wraddress 0 0 10 0
// Retrieval info: CONNECT: @rdaddress 0 0 10 0 rdaddress 0 0 10 0
// Retrieval info: CONNECT: @wren 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: @inclock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @aclr 0 0 0 0 aclr 0 0 0 0
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
```

---

```
/*-------------------------------------------------------*/
/*
(c) 2004 B.R. Phelps, North Carolina State University
Module test_Full5.v.v

Created by: Brian R. Phelps   12/22/03

Project: Sliding Window Packet Switch FPGA Implementation
                  from Dr. Sanjeev Kumar's packet switch algorithm.

Module Description:
```

```
            Test Fixture for whole Project

*/
/*-------------------------------------------------------*/

`timescale 1ns/100ps
`include "SW_Top_Level.v"
`include "../PAC/PAC_Full.v"
`include "../PAC/PAC_Header.v"
`include "../PAC/PAC_Memory.v"
`include "../PAC/PAC_NCFSM.v"
`include "../Mega_Functions/ram_pac.v"
`include "../Mega_Functions/shift_reg2.v"
`include "../Mega_Functions/MEM_MOD.v"
`include "../PAC/PAC_SWCounter.v"
`include "../PAC/PAC_600.v"
`include "../PAC/PAC_650.v"
`include "../IIN/IIN_CB.v"
`include "../IIN/IIN_FSM.v"
`include "../IIN/IIN_H.v"
`include "../IIN/IIN.v"
`include "../MM/MM.v"
`include "../MM/MM_RAM.v"
`include "../MM/MM_FSM.v"
`include "../OIN/OIN.v"
`include "../PS/PS.v"

//    To run for a long time on Load Sharing Facility
// ssh -l <login> -X lsf.ncsu.edu
//    add lsf
//    bsub -n 4,10 make test5

/*-------------------------------------------------------------------------
---*/
/* This File reads the port input from a dat file (generated by test_sw.m)
           and creates the packet from a constant.*/
/*-------------------------------------------------------------------------
---*/
module test_bench;

integer k,kk, fid1, fid2, fid3;

reg clock, reset;
reg[32:0] w_index;
reg[8:0] b_index;

reg[8:0]counter;


wire[3:0] ser_in;                   /* serial data input*/

reg  [2:0]port [0:2000000];
reg  [3:0]mem [0:40000];

wire[511:0]mem_bits0, mem_bits1, mem_bits2, mem_bits3;
```

```
wire[3:0] PS1_pkt_here_out;

wire        PS1_port1_out,
                  PS1_port2_out,
                  PS1_port3_out,
                  PS1_port4_out;

initial
      begin
            fid1=$fopen("verilog_test_output.dat");   // Output file for
comparison to matlab
            fid3=$fopen("verilog_d_i.dat");     // File that records when
a duplicate i is assigned

            //fid2=$fopen("verilog5_debug.dat");
            $readmemh("../../verilog_port.dat",port); // Input Port File

         //$dumpfile("test.vcd"); // save waveforms in this file (MUST 1st
set cycle count to a low value!!!)
            //$dumpvars;

            #0 k = 0; clock = 0; reset = 0;
            #10 reset=1;

            // #ns to run = (number cycles)/(10*512)
            //#1600000
            //#120000
            #1286597000 // 2,400,000 => 470 cycles

            #40
            $system("zwrite brphelps -m Simulation Done");  // Tell me
when it is finished
            $fclose(fid1); $fclose(fid3); //$fclose(fid2);
            $finish;
      end

// the different packets to generate (injects the port number)
assign
mem_bits0={508'hbeefead1beefead2beefead3beefead4beefead5beefead6beefead7be
efead8beefead9beefeadabeefeadbbeefeadcbeefeaddbeefeadebeefeadfbfff000,1'b0
,port[w_index]};
assign
mem_bits1={508'hbeefead1beefead2beefead3beefead4beefead5beefead6beefead7be
efead8beefead9beefeadabeefeadbbeefeadcbeefeaddbeefeadebeefeadfbfff000,1'b0
,port[w_index+1]};
assign
mem_bits2={508'hbeefead1beefead2beefead3beefead4beefead5beefead6beefead7be
efead8beefead9beefeadabeefeadbbeefeadcbeefeaddbeefeadebeefeadfbfff000,1'b0
,port[w_index+2]};
assign
mem_bits3={508'hbeefead1beefead2beefead3beefead4beefead5beefead6beefead7be
efead8beefead9beefeadabeefeadbbeefeadcbeefeaddbeefeadebeefeadfbfff000,1'b0
,port[w_index+3]};

// read in the packets to PAC serially
assign ser_in[0] = mem_bits0[b_index];
assign ser_in[1] = mem_bits1[b_index];
```

```
assign ser_in[2] = mem_bits2[b_index];
assign ser_in[3] = mem_bits3[b_index];

always@(posedge clock)
      begin
            if (!reset)
                  begin
                        w_index <= 0;
                        b_index <= 0;  k <= -4; kk <= 0;
                        counter<=0;
                  end
            else
                  begin

/*-------------DEBUG-----------------------------*/
                        if (SW.IIN1.i_dupl_out)
                                    $fwrite(fid3,"------------------i
duplicated Cycle: %d------------:\n",kk+2);
/*-------------DEBUG-----------------------------*/


                        // These indexes are word and bit indexes for
reading the port file
                        b_index <= b_index + 1;
                        w_index <= w_index + 4*(b_index == 9'd511);
                        // The Counter is used for deciding when a packet
is ready
                        counter <= counter+1'b1;
                        if ( (counter==8'd101) )
                              begin
                                    kk=kk+1;

                                    //  Write output packets to file for
comparison
                                    // Commented out to save space (quota
space)
                                    //$fwrite(fid1,"Cycle: %d\nPackets on
port Output:\n",kk+2);
                                    //$display("Cycle: %d:\n",kk+2);
                                    // Write i/j/k/d to file
                                    if(PS1_pkt_here_out[0])

      $fwrite(fid1,"%d/%d/%d/%d\n",s1.q[12:9],s1.q[8:5],s1.q[4:3],s1.q[2:0
]);
                                    else
                                          $fwrite(fid1,"0/0/0/0\n");

                                    if(PS1_pkt_here_out[1])

      $fwrite(fid1,"%d/%d/%d/%d\n",s2.q[12:9],s2.q[8:5],s2.q[4:3],s2.q[2:0
]);
                                    else
                                          $fwrite(fid1,"0/0/0/0\n");

                                    if(PS1_pkt_here_out[2])
```

```
      $fwrite(fid1,"%d/%d/%d/%d\n",s3.q[12:9],s3.q[8:5],s3.q[4:3],s3.q[2:0
]);
                                    else
                                          $fwrite(fid1,"0/0/0/0\n");

                                    if(PS1_pkt_here_out[3])

      $fwrite(fid1,"%d/%d/%d/%d\n\n",s4.q[12:9],s4.q[8:5],s4.q[4:3],s4.q[2
:0]);
                                    else
                                          $fwrite(fid1,"0/0/0/0\n\n");

                               end

                  end
      end

always #5 clock = ~clock;



/*------------------------- Top level module connections    --------------
----------------*/
/*
                  inputs:
                        clock, reset,
                        ser_in[3:0]             // serial packet input


                  outputs:
                        PS1_port1_out           // serial packet output
                        PS1_port2_out
                        PS1_port3_out
                        PS1_port4_out

                        PS1_pkt_here_out[3:0]   // packet present(no
delay)
*/

SW_Top_Level SW(.clock(clock), .reset(reset), .ser_in(ser_in),
                        .PS1_port1_out(PS1_port1_out),
                        .PS1_port2_out(PS1_port2_out),
                        .PS1_port3_out(PS1_port3_out),
                        .PS1_port4_out(PS1_port4_out),
                        .PS1_pkt_here_out(PS1_pkt_here_out) );



/*------------------------------------------------------------------------
------------*/

/*--------------Parallelize the serial data for easy viewing--------------
-------------*/

// These are Mega functions From Quartus II software, but are not
synthesized here!
```

```
//          They are just used to parallelize the data for viewing in the
test fixture

shift_reg2 s1(.clock(clock && PS1_pkt_here_out[0]), .sclr(!reset),
  .shiftin(PS1_port1_out),
      .q() );

shift_reg2 s2(.clock(clock && PS1_pkt_here_out[1]), .sclr(!reset),
  .shiftin(PS1_port2_out),
      .q() );

shift_reg2 s3(.clock(clock && PS1_pkt_here_out[2]), .sclr(!reset),
  .shiftin(PS1_port3_out),
      .q() );

shift_reg2 s4(.clock(clock && PS1_pkt_here_out[3]), .sclr(!reset),
  .shiftin(PS1_port4_out),
      .q() );
/*-------------------------------------------------------------------------
---------------*/
endmodule
```

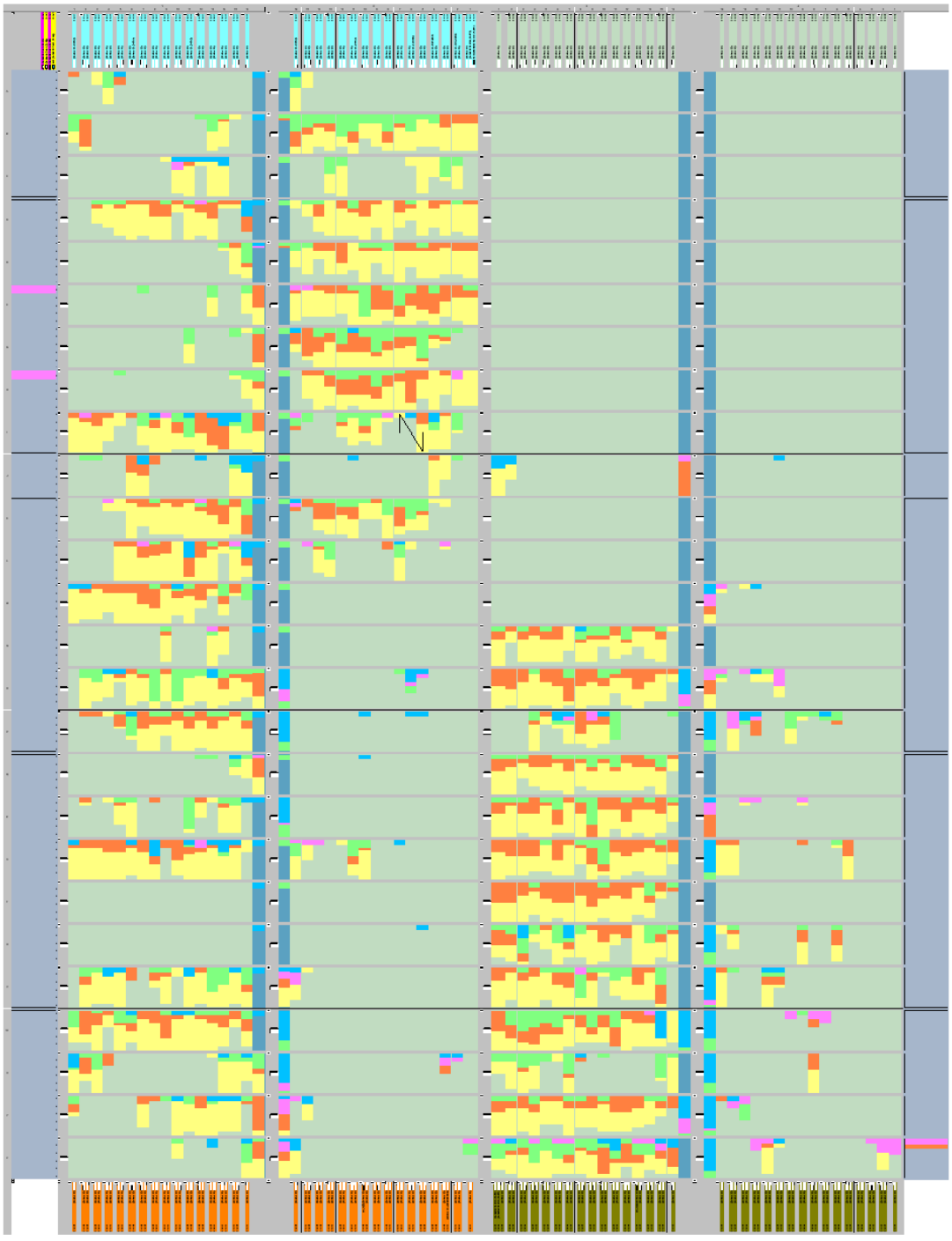
---

```
# Makefile For whole Verilog Project Simulation

test  :
      verilog test_Full5.v +define+NO_PLI ../Mega_Functions/altera_mf.v
+define+NO_PLI ../Mega_Functions/220model.v


clean :
      rm -f *.trn;
      rm -f *.vcd;
      rm -f *.dsn;
```

---

## Synthesis Results For The SW Switch
(floorplan of FPGA)

Date: March 26, 2004 Project: project_megafunctions



Page 1 of

```
Resource    Usage
Logic cells 4,648 / 16,640 ( 27 % )
Registers   1,101 / 19,536 ( 5 % )
User inserted logic cells     0
I/O pins    10 / 488 ( 2 % )
    -- Clock pins       0
    -- Dedicated input pins   0 / 4 ( 0 % )
Global signals    2
ESBs  64 / 104 ( 61 % )
Macrocells  0 / 1,664 ( 0 % )
ESB pterm bits used     0 / 212,992 ( 0 % )
ESB CAM bits used 0 / 212,992 ( 0 % )
Total memory bits 69,632 / 212,992 ( 32 % )
Total RAM block bits    131,072 / 212,992 ( 61 % )
FastRow interconnects   0 / 120 ( 0 % )
LVDS transmitters 0 / 16 ( 0 % )
LVDS receivers    0 / 16 ( 0 % )
Maximum fan-out node    clock
Maximum fan-out   1677
Total fan-out     25424
Average fan-out   4.86
```

```
---------------------------------
; Analysis & Synthesis Messages  ;
---------------------------------
Info: *******************************************************************
Info: Running Quartus II Analysis & Synthesis
  Info: Version 3.0 Build 223 08/14/2003 Service Pack 1 SJ Full Version
  Info: Processing started: Tue Mar 09 15:13:02 2004
Info: Command: quartus_map --import_settings_files=on --export_settings_files=off
project_megafunctions -c Project_Megafunctions
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\OIN\OIN.v
  Info: Found entity 1: OIN
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PS\PS.v
  Info: Found entity 1: PS
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\MM\MM.v
  Info: Found entity 1: MM
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\MM\MM_FSM.v
  Info: Found entity 1: MM_FSM
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\MM\MM_RAM.v
  Info: Found entity 1: MM_RAM
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\IIN\IIN.v
  Info: Found entity 1: IIN
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\IIN\IIN_CB.v
  Info: Found entity 1: IIN_CB
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\IIN\IIN_FSM.v
  Info: Found entity 1: IIN_FSM
```

```
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\IIN\IIN_H.v
  Info: Found entity 1: IIN_H
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_650.v
  Info: Found entity 1: PAC_650
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_Full.v
  Info: Found entity 1: PAC_Full
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_Header.v
  Info: Found entity 1: PAC_Header
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_Memory.v
  Info: Found entity 1: PAC_Memory
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_NCFSM.v
  Info: Found entity 1: PAC_NCFSM
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_SWCounter.v
  Info: Found entity 1: PAC_SWCounter
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project\PAC\PAC_600.v
  Info: Found entity 1: PAC_600
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\MEM_MOD.v
  Info: Found entity 1: MEM_MOD
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\ram_pac.v
  Info: Found entity 1: ram_pac
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\shift_reg2.v
  Info: Found entity 1: shift_reg2
Info: Found 1 design units and 1 entities in source file C:\Documents and
Settings\brphelps\My Documents\Quartus_Projects\SW_2\Project_Megafunctions.v
  Info: Found entity 1: project_megafunctions
Info: Found 1 design units and 1 entities in source file
c:\quartus\libraries\megafunctions\lpm_shiftreg.tdf
  Info: Found entity 1: lpm_shiftreg
Info: Found 1 design units and 1 entities in source file
c:\quartus\libraries\megafunctions\altdpram.tdf
  Info: Found entity 1: altdpram
Warning: Verilog HDL expression warning at MM_RAM.v(46): truncated operand with size 32 to
match size of smaller operand (4)
Info: Inferred 12 megafunctions from design logic
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=6) from the following logic:
OIN:OIN1|counter_rtl_873
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M8|w_addr_reg_rtl_874
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=11) from the following logic:
MM:MM1|MM_FSM:MFSM|counter_rtl_875
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M7|w_addr_reg_rtl_876
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M6|w_addr_reg_rtl_877
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M5|w_addr_reg_rtl_878
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M4|w_addr_reg_rtl_879
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M3|w_addr_reg_rtl_880
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M2|w_addr_reg_rtl_881
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=7) from the following logic:
MM:MM1|MM_RAM:M1|w_addr_reg_rtl_882
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=9) from the following logic:
IIN:IIN1|IIN_FSM:FSM|counter_rtl_883
  Info: Inferred lpm_counter megafunction (LPM_WIDTH=9) from the following logic:
PAC_Full:PAC|PAC_NCFSM:NCFSM|counter_rtl_884
Info: Found 1 design units and 1 entities in source file
c:\quartus\libraries\megafunctions\lpm_counter.tdf
```

```
  Info: Found entity 1: lpm_counter
Info: Found 1 design units and 1 entities in source file
c:\quartus\libraries\megafunctions\alt_synch_counter.tdf
  Info: Found entity 1: alt_synch_counter
Info: Implemented 5234 device resources after synthesis - the final resource count might be
different
  Info: Implemented 6 input pins
  Info: Implemented 4 output pins
  Info: Implemented 4648 logic cells
  Info: Implemented 576 RAM segments
Info: Quartus II Analysis & Synthesis was successful. 0 errors, 1 warning
  Info: Processing ended: Tue Mar 09 15:19:49 2004
  Info: Elapsed time: 00:06:46
Info: Writing report file Project_Megafunctions.map.rpt
```

---

```
/------------------------------/
; Timing Analyzer Summary  ;
/------------------------/
Type  Slack Required Time     Actual Time Source Name Destination Name
Clock Setup: 'clock'    N/A   None  22.14 MHz ( period = 45.165 ns )

Worst-case tsu    N/A   None  13.949 ns
Worst-case tco    N/A   None  49.900 ns
Worst-case th     N/A   None  0.411 ns
Worst-case minimum tco  N/A   None  13.640 ns
```

---

```
-----------------------------
; Timing Analyzer Messages  ;
-----------------------------
Info: ********************************************************************
Info: Running Quartus II Timing Analyzer
  Info: Version 3.0 Build 223 08/14/2003 Service Pack 1 SJ Full Version
  Info: Processing started: Tue Mar 09 15:50:01 2004
Info: Command: quartus_tan --import_settings_files=off --export_settings_files=off
project_megafunctions -c Project_Megafunctions
Warning: Found pins functioning as undefined clocks and/or memory enables
  Info: Assuming node clock is an undefined clock
Info: Clock clock has Internal fmax of 22.14 MHz between source register
PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_synch_counter:wysi_counter|sload
_path[6] and destination register PAC_Full:PAC|PAC_600:P_600|Qd[4][2] (period= 45.165 ns)

  Info: + Longest register to register delay is 44.596 ns
    Info: 1: + IC(0.000 ns) + CELL(0.219 ns) = 0.219 ns; Loc. = LC7_11_Z3; REG Node =
'PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_synch_counter:wysi_counter|sloa
d_path[6]'
    Info: 2: + IC(1.136 ns) + CELL(1.207 ns) = 2.562 ns; Loc. = LC8_10_Z3; COMB Node =
'PAC_Full:PAC|PAC_NCFSM:NCFSM|reduce_nor_30~117'
    Info: 3: + IC(0.300 ns) + CELL(1.207 ns) = 4.069 ns; Loc. = LC2_10_Z3; COMB Node =
'PAC_Full:PAC|PAC_NCFSM:NCFSM|i~2'
    Info: 4: + IC(0.324 ns) + CELL(1.192 ns) = 5.585 ns; Loc. = LC4_9_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|Select_112_rtl_61~0'
    Info: 5: + IC(0.294 ns) + CELL(0.486 ns) = 6.365 ns; Loc. = LC2_9_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|Select_112_rtl_61~1'
    Info: 6: + IC(1.315 ns) + CELL(1.082 ns) = 8.762 ns; Loc. = LC2_1_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|Mux_5_rtl_2172~0'
    Info: 7: + IC(0.287 ns) + CELL(0.486 ns) = 9.535 ns; Loc. = LC1_1_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|Mux_5_rtl_2172~1'
    Info: 8: + IC(0.287 ns) + CELL(0.486 ns) = 10.308 ns; Loc. = LC10_1_Z3; COMB Node =
'rtl~2160'
    Info: 9: + IC(2.367 ns) + CELL(1.521 ns) = 14.196 ns; Loc. = LC6_2_W3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|add_11~4COUT'
    Info: 10: + IC(0.000 ns) + CELL(1.000 ns) = 15.196 ns; Loc. = LC7_2_W3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|add_11~5'
    Info: 11: + IC(0.273 ns) + CELL(1.082 ns) = 16.551 ns; Loc. = LC9_2_W3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|reduce_nor_62~12'
```

```
    Info: 12: + IC(0.283 ns) + CELL(1.082 ns) = 17.916 ns; Loc. = LC6_3_W3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|j_f[1]~25'
    Info: 13: + IC(0.287 ns) + CELL(1.192 ns) = 19.395 ns; Loc. = LC4_3_W3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~567'
    Info: 14: + IC(0.297 ns) + CELL(1.082 ns) = 20.774 ns; Loc. = LC9_3_W3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|j_524_addr[1]~1'
    Info: 15: + IC(2.636 ns) + CELL(1.082 ns) = 24.492 ns; Loc. = LC10_2_R3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|Mux_137_rtl_407_rtl_503_rtl_599_rtl_1816~0'
    Info: 16: + IC(0.273 ns) + CELL(0.486 ns) = 25.251 ns; Loc. = LC9_1_R3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|Mux_137_rtl_407_rtl_503_rtl_599_rtl_1816~1'
    Info: 17: + IC(0.259 ns) + CELL(1.082 ns) = 26.592 ns; Loc. = LC5_1_R3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|reduce_nor_139~28'
    Info: 18: + IC(2.626 ns) + CELL(1.207 ns) = 30.425 ns; Loc. = LC7_11_U3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|reduce_nor_139~130'
    Info: 19: + IC(1.162 ns) + CELL(1.082 ns) = 32.669 ns; Loc. = LC4_10_U3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|add_144~0'
    Info: 20: + IC(1.210 ns) + CELL(1.082 ns) = 34.961 ns; Loc. = LC1_9_U3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|add_145~57'
    Info: 21: + IC(0.273 ns) + CELL(1.207 ns) = 36.441 ns; Loc. = LC5_9_U3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|i~52324'
    Info: 22: + IC(0.304 ns) + CELL(1.192 ns) = 37.937 ns; Loc. = LC9_8_U3; COMB Node =
'PAC_Full:PAC|PAC_650:P_650|add_55~15'
    Info: 23: + IC(0.297 ns) + CELL(1.082 ns) = 39.316 ns; Loc. = LC3_8_U3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|LessThan_101_rtl_607~19'
    Info: 24: + IC(0.269 ns) + CELL(1.207 ns) = 40.792 ns; Loc. = LC8_8_U3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~1'
    Info: 25: + IC(2.748 ns) + CELL(0.486 ns) = 44.026 ns; Loc. = LC7_14_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~1217'
    Info: 26: + IC(0.310 ns) + CELL(0.260 ns) = 44.596 ns; Loc. = LC6_14_Z3; REG Node =
'PAC_Full:PAC|PAC_600:P_600|Qd[4][2]'
    Info: Total cell delay = 24.779 ns
    Info: Total interconnect delay = 19.817 ns
  Info: - Smallest clock skew is 0.000 ns
    Info: + Shortest clock path from clock clock to destination register is 3.878 ns
      Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
      Info: 2: + IC(1.977 ns) + CELL(0.000 ns) = 3.878 ns; Loc. = LC6_14_Z3; REG Node =
'PAC_Full:PAC|PAC_600:P_600|Qd[4][2]'
      Info: Total cell delay = 1.901 ns
      Info: Total interconnect delay = 1.977 ns
    Info: - Longest clock path from clock clock to source register is 3.878 ns
      Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
      Info: 2: + IC(1.977 ns) + CELL(0.000 ns) = 3.878 ns; Loc. = LC7_11_Z3; REG Node =
'PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_synch_counter:wysi_counter|sloa
d_path[6]'
      Info: Total cell delay = 1.901 ns
      Info: Total interconnect delay = 1.977 ns
  Info: + Micro clock to output delay of source is 0.454 ns
  Info: + Micro setup delay of destination is 0.115 ns
Info: tsu for register PAC_Full:PAC|PAC_600:P_600|Qd[2][3] (data pin = reset, clock pin =
clock) is 13.949 ns
  Info: + Longest pin to register delay is 17.712 ns
    Info: 1: + IC(0.000 ns) + CELL(1.980 ns) = 1.980 ns; Loc. = Pin_H14; PIN Node = 'reset'
    Info: 2: + IC(1.891 ns) + CELL(1.207 ns) = 5.078 ns; Loc. = LC6_15_U3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~8024'
    Info: 3: + IC(2.363 ns) + CELL(1.207 ns) = 8.648 ns; Loc. = LC2_15_Z3; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~8042'
    Info: 4: + IC(3.072 ns) + CELL(1.207 ns) = 12.927 ns; Loc. = LC5_1_Z4; COMB Node =
'PAC_Full:PAC|PAC_600:P_600|i~8597'
    Info: 5: + IC(3.838 ns) + CELL(0.947 ns) = 17.712 ns; Loc. = LC8_4_Z3; REG Node =
'PAC_Full:PAC|PAC_600:P_600|Qd[2][3]'
    Info: Total cell delay = 6.548 ns
    Info: Total interconnect delay = 11.164 ns
  Info: + Micro setup delay of destination is 0.115 ns
  Info: - Shortest clock path from clock clock to destination register is 3.878 ns
    Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
    Info: 2: + IC(1.977 ns) + CELL(0.000 ns) = 3.878 ns; Loc. = LC8_4_Z3; REG Node =
'PAC_Full:PAC|PAC_600:P_600|Qd[2][3]'
    Info: Total cell delay = 1.901 ns
    Info: Total interconnect delay = 1.977 ns
Info: tco from clock clock to destination pin PS1_port1_out through register
MM:MM1|PAC_SWCounter:SWC|SW_j[1]~reg0 is 49.900 ns
```

```
  Info: + Longest clock path from clock clock to source register is 3.857 ns
    Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
    Info: 2: + IC(1.956 ns) + CELL(0.000 ns) = 3.857 ns; Loc. = LC8_10_I1; REG Node =
'MM:MM1|PAC_SWCounter:SWC|SW_j[1]~reg0'
    Info: Total cell delay = 1.901 ns
    Info: Total interconnect delay = 1.956 ns
  Info: + Micro clock to output delay of source is 0.454 ns
  Info: + Longest register to pin delay is 45.589 ns
    Info: 1: + IC(0.000 ns) + CELL(0.219 ns) = 0.219 ns; Loc. = LC8_10_I1; REG Node =
'MM:MM1|PAC_SWCounter:SWC|SW_j[1]~reg0'
    Info: 2: + IC(3.695 ns) + CELL(1.082 ns) = 4.996 ns; Loc. = LC3_14_P1; COMB Node =
'MM:MM1|MM_RAM:M1|Mux_75_rtl_951~4'
    Info: 3: + IC(0.280 ns) + CELL(0.486 ns) = 5.762 ns; Loc. = LC8_14_P1; COMB Node =
'MM:MM1|MM_RAM:M1|Mux_75_rtl_951~9'
    Info: 4: + IC(1.197 ns) + CELL(1.082 ns) = 8.041 ns; Loc. = LC3_10_P1; COMB Node =
'MM:MM1|MM_RAM:M1|Mux_75_rtl_950~0'
    Info: 5: + IC(0.280 ns) + CELL(0.486 ns) = 8.807 ns; Loc. = LC10_9_P1; COMB Node =
'MM:MM1|MM_RAM:M1|Mux_75_rtl_950~1'
    Info: 6: + IC(3.368 ns) + CELL(1.082 ns) = 13.257 ns; Loc. = LC8_16_L1; COMB Node =
'MM:MM1|MM_RAM:M1|reduce_nor_79'
    Info: 7: + IC(4.464 ns) + CELL(0.486 ns) = 18.207 ns; Loc. = LC3_13_B2; COMB Node =
'OIN:OIN1|dd1_f[2]~1'
    Info: 8: + IC(1.101 ns) + CELL(1.207 ns) = 20.515 ns; Loc. = LC9_12_B2; COMB Node =
'OIN:OIN1|Decoder_71~482'
    Info: 9: + IC(2.706 ns) + CELL(1.082 ns) = 24.303 ns; Loc. = LC8_6_G2; COMB Node =
'OIN:OIN1|Select_132_rtl_68~24'
    Info: 10: + IC(0.266 ns) + CELL(1.207 ns) = 25.776 ns; Loc. = LC8_7_G2; COMB Node =
'OIN:OIN1|Select_132_rtl_68~20'
    Info: 11: + IC(2.659 ns) + CELL(1.082 ns) = 29.517 ns; Loc. = LC1_13_B2; COMB Node =
'OIN:OIN1|Select_134_rtl_70~56'
    Info: 12: + IC(2.544 ns) + CELL(0.486 ns) = 32.547 ns; Loc. = LC7_12_F2; COMB Node =
'OIN:OIN1|Select_134_rtl_70~72'
    Info: 13: + IC(0.280 ns) + CELL(1.192 ns) = 34.019 ns; Loc. = LC9_11_F2; COMB Node =
'PS:PS1|ps1f[0]~10'
    Info: 14: + IC(2.531 ns) + CELL(0.486 ns) = 37.036 ns; Loc. = LC6_2_H2; COMB Node =
'PS:PS1|Select_77_rtl_24_rtl_1974~0'
    Info: 15: + IC(0.266 ns) + CELL(1.082 ns) = 38.384 ns; Loc. = LC5_3_H2; COMB Node =
'PS:PS1|Select_77_rtl_24_rtl_1974~1'
    Info: 16: + IC(0.266 ns) + CELL(0.486 ns) = 39.136 ns; Loc. = LC4_2_H2; COMB Node =
'rtl~732'
    Info: 17: + IC(3.742 ns) + CELL(2.711 ns) = 45.589 ns; Loc. = Pin_L7; PIN Node =
'PS1_port1_out'
    Info: Total cell delay = 15.944 ns
    Info: Total interconnect delay = 29.645 ns
Info: th for register OIN:OIN1|dd1[1] (data pin = reset, clock pin = clock) is 0.411 ns
  Info: + Longest clock path from clock clock to destination register is 3.878 ns
    Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
    Info: 2: + IC(1.977 ns) + CELL(0.000 ns) = 3.878 ns; Loc. = LC2_16_B2; REG Node =
'OIN:OIN1|dd1[1]'
    Info: Total cell delay = 1.901 ns
    Info: Total interconnect delay = 1.977 ns
  Info: + Micro hold delay of destination is 0.356 ns
  Info: - Shortest pin to register delay is 3.823 ns
    Info: 1: + IC(0.000 ns) + CELL(1.980 ns) = 1.980 ns; Loc. = Pin_H14; PIN Node = 'reset'
    Info: 2: + IC(1.583 ns) + CELL(0.260 ns) = 3.823 ns; Loc. = LC2_16_B2; REG Node =
'OIN:OIN1|dd1[1]'
    Info: Total cell delay = 2.240 ns
    Info: Total interconnect delay = 1.583 ns
Info: Minimum tco from clock clock to destination pin PS1_port2_out through register
PS:PS1|counter[2] is 13.640 ns
  Info: + Shortest clock path from clock clock to source register is 3.875 ns
    Info: 1: + IC(0.000 ns) + CELL(1.901 ns) = 1.901 ns; Loc. = Pin_R6; CLK Node = 'clock'
    Info: 2: + IC(1.974 ns) + CELL(0.000 ns) = 3.875 ns; Loc. = LC10_3_C2; REG Node =
'PS:PS1|counter[2]'
    Info: Total cell delay = 1.901 ns
    Info: Total interconnect delay = 1.974 ns
  Info: + Micro clock to output delay of source is 0.454 ns
  Info: + Shortest register to pin delay is 9.311 ns
    Info: 1: + IC(0.000 ns) + CELL(0.219 ns) = 0.219 ns; Loc. = LC10_3_C2; REG Node =
'PS:PS1|counter[2]'
```

```
    Info: 2: + IC(2.837 ns) + CELL(0.486 ns) = 3.542 ns; Loc. = LC7_15_F2; COMB Node =
'rtl~759'
    Info: 3: + IC(3.058 ns) + CELL(2.711 ns) = 9.311 ns; Loc. = Pin_K5; PIN Node =
'PS1_port2_out'
    Info: Total cell delay = 3.416 ns
    Info: Total interconnect delay = 5.895 ns
Info: Quartus II Timing Analyzer was successful. 0 errors, 1 warning
  Info: Processing ended: Tue Mar 09 15:51:19 2004
  Info: Elapsed time: 00:01:17
Info: Writing report file Project_Megafunctions.tan.rpt
```

---

```
/------------------------------/
Resource usage by module
/------------------------------/

Compilation Hierarchy Node    Logic Cells    Registers      Memory Bits    Pins    Virtual
Pins    LUT-Only LCs   Register-Only LCs     LUT/Register LCs       Full Hierarchy Name
|project_megafunctions 4648 (126)     1101    8256    10      0       3547 (126)     0 (0)
       1101 (0)       |project_megafunctions
   |IIN:IIN1|  849 (0) 49      32      0       0       800 (0) 0 (0)   49 (0)
        |project_megafunctions|IIN:IIN1
      |IIN_CB:CB1|     76 (76) 0       0       0       0       76 (76) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB1
      |IIN_CB:CB2|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB2
      |IIN_CB:CB3|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB3
      |IIN_CB:CB4|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB4
      |IIN_CB:CB5|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB5
      |IIN_CB:CB6|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB6
      |IIN_CB:CB7|     73 (73) 0       0       0       0       73 (73) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB7
      |IIN_CB:CB8|     74 (74) 0       0       0       0       74 (74) 0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_CB:CB8
      |IIN_FSM:FSM|    74 (65) 9       0       0       0       65 (65) 0 (0)   9 (0)
        |project_megafunctions|IIN:IIN1|IIN_FSM:FSM
          |lpm_counter:counter_rtl_883|       9 (0)   9       0       0       0       0 (0)   0
(0)     9 (0)   |project_megafunctions|IIN:IIN1|IIN_FSM:FSM|lpm_counter:counter_rtl_883
             |alt_synch_counter:wysi_counter| 9 (9)   9       0       0       0       0 (0)   0
(0)     9 (9)
        |project_megafunctions|IIN:IIN1|IIN_FSM:FSM|lpm_counter:counter_rtl_883|alt_synch_coun
ter:wysi_counter
      |IIN_H:H1|       48 (48) 10      8       0       0       38 (38) 0 (0)   10 (10)
        |project_megafunctions|IIN:IIN1|IIN_H:H1
          |ram_pac:ram2|       0 (0)   0       8       0       0       0 (0)   0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H1|ram_pac:ram2
             |altdpram:altdpram_component|    0 (0)   0       8       0       0       0 (0)   0
(0)     0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H1|ram_pac:ram2|altdpram:altdpram_component
      |IIN_H:H2|       47 (46) 10      8       0       0       37 (36) 0 (0)   10 (10)
        |project_megafunctions|IIN:IIN1|IIN_H:H2
          |ram_pac:ram2|       1 (0)   0       8       0       0       1 (0)   0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H2|ram_pac:ram2
             |altdpram:altdpram_component|    1 (1)   0       8       0       0       1 (1)   0
(0)     0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H2|ram_pac:ram2|altdpram:altdpram_component
      |IIN_H:H3|       46 (46) 10      8       0       0       36 (36) 0 (0)   10 (10)
        |project_megafunctions|IIN:IIN1|IIN_H:H3
          |ram_pac:ram2|       0 (0)   0       8       0       0       0 (0)   0 (0)   0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H3|ram_pac:ram2
             |altdpram:altdpram_component|    0 (0)   0       8       0       0       0 (0)   0
(0)     0 (0)
        |project_megafunctions|IIN:IIN1|IIN_H:H3|ram_pac:ram2|altdpram:altdpram_component
      |IIN_H:H4|       46 (46) 10      8       0       0       36 (36) 0 (0)   10 (10)
        |project_megafunctions|IIN:IIN1|IIN_H:H4
```

```
        |ram_pac:ram2|       0 (0)   0       8       0       0       0 (0)   0 (0)   0 (0)
       |project_megafunctions|IIN:IIN1|IIN_H:H4|ram_pac:ram2
           |altdpram:altdpram_component|    0 (0)   0       8       0       0       0 (0)   0
(0)    0 (0)
       |project_megafunctions|IIN:IIN1|IIN_H:H4|ram_pac:ram2|altdpram:altdpram_component
   |MM:MM1|    972 (0) 322     8192    0       0       650 (0) 0 (0)   322 (0)
       |project_megafunctions|MM:MM1
      |MM_FSM:MFSM|    18 (7)  12      0       0       0       6 (6)   0 (0)   12 (1)
       |project_megafunctions|MM:MM1|MM_FSM:MFSM
        |lpm_counter:counter_rtl_875|       11 (0)  11      0       0       0       0 (0)   0
(0)    11 (0)  |project_megafunctions|MM:MM1|MM_FSM:MFSM|lpm_counter:counter_rtl_875
           |alt_synch_counter:wysi_counter| 11 (11) 11      0       0       0       0 (0)   0
(0)    11 (11)
       |project_megafunctions|MM:MM1|MM_FSM:MFSM|lpm_counter:counter_rtl_875|alt_synch_counte
r:wysi_counter
      |MM_RAM:M1|      117 (109)      38      1024    0       0       79 (78) 0 (0)   38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M1
        |MEM_MOD:MM_ram|     1 (0)   0       1024    0       0       1 (0)   0 (0)   0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M1|MEM_MOD:MM_ram
           |altdpram:altdpram_component|    1 (1)   0       1024    0       0       1 (1)   0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M1|MEM_MOD:MM_ram|altdpram:altdpram_component
        |lpm_counter:w_addr_reg_rtl_882|    7 (0)   7       0       0       0       0 (0)   0
(0)    7 (0)   |project_megafunctions|MM:MM1|MM_RAM:M1|lpm_counter:w_addr_reg_rtl_882
           |alt_synch_counter:wysi_counter| 7 (7)   7       0       0       0       0 (0)   0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M1|lpm_counter:w_addr_reg_rtl_882|alt_synch_count
er:wysi_counter
      |MM_RAM:M2|      116 (109)      38      1024    0       0       78 (78) 0 (0)   38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M2
        |MEM_MOD:MM_ram|     0 (0)   0       1024    0       0       0 (0)   0 (0)   0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M2|MEM_MOD:MM_ram
           |altdpram:altdpram_component|    0 (0)   0       1024    0       0       0 (0)   0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M2|MEM_MOD:MM_ram|altdpram:altdpram_component
        |lpm_counter:w_addr_reg_rtl_881|    7 (0)   7       0       0       0       0 (0)   0
(0)    7 (0)   |project_megafunctions|MM:MM1|MM_RAM:M2|lpm_counter:w_addr_reg_rtl_881
           |alt_synch_counter:wysi_counter| 7 (7)   7       0       0       0       0 (0)   0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M2|lpm_counter:w_addr_reg_rtl_881|alt_synch_count
er:wysi_counter
      |MM_RAM:M3|      116 (109)      38      1024    0       0       78 (78) 0 (0)   38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M3
        |MEM_MOD:MM_ram|     0 (0)   0       1024    0       0       0 (0)   0 (0)   0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M3|MEM_MOD:MM_ram
           |altdpram:altdpram_component|    0 (0)   0       1024    0       0       0 (0)   0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M3|MEM_MOD:MM_ram|altdpram:altdpram_component
        |lpm_counter:w_addr_reg_rtl_880|    7 (0)   7       0       0       0       0 (0)   0
(0)    7 (0)   |project_megafunctions|MM:MM1|MM_RAM:M3|lpm_counter:w_addr_reg_rtl_880
           |alt_synch_counter:wysi_counter| 7 (7)   7       0       0       0       0 (0)   0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M3|lpm_counter:w_addr_reg_rtl_880|alt_synch_count
er:wysi_counter
      |MM_RAM:M4|      116 (109)      38      1024    0       0       78 (78) 0 (0)   38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M4
        |MEM_MOD:MM_ram|     0 (0)   0       1024    0       0       0 (0)   0 (0)   0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M4|MEM_MOD:MM_ram
           |altdpram:altdpram_component|    0 (0)   0       1024    0       0       0 (0)   0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M4|MEM_MOD:MM_ram|altdpram:altdpram_component
        |lpm_counter:w_addr_reg_rtl_879|    7 (0)   7       0       0       0       0 (0)   0
(0)    7 (0)   |project_megafunctions|MM:MM1|MM_RAM:M4|lpm_counter:w_addr_reg_rtl_879
           |alt_synch_counter:wysi_counter| 7 (7)   7       0       0       0       0 (0)   0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M4|lpm_counter:w_addr_reg_rtl_879|alt_synch_count
er:wysi_counter
      |MM_RAM:M5|      116 (109)      38      1024    0       0       78 (78) 0 (0)   38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M5
        |MEM_MOD:MM_ram|     0 (0)   0       1024    0       0       0 (0)   0 (0)   0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M5|MEM_MOD:MM_ram
```

```
            |altdpram:altdpram_component|    0 (0)  0      1024   0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M5|MEM_MOD:MM_ram|altdpram:altdpram_component
         |lpm_counter:w_addr_reg_rtl_878|    7 (0)  7      0      0      0      0 (0)  0
(0)    7 (0)  |project_megafunctions|MM:MM1|MM_RAM:M5|lpm_counter:w_addr_reg_rtl_878
            |alt_synch_counter:wysi_counter| 7 (7)  7      0      0      0      0 (0)  0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M5|lpm_counter:w_addr_reg_rtl_878|alt_synch_count
er:wysi_counter
      |MM_RAM:M6|     116 (109)      38     1024   0      0      78 (78)0 (0)  38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M6
         |MEM_MOD:MM_ram|     0 (0)  0      1024   0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M6|MEM_MOD:MM_ram
            |altdpram:altdpram_component|    0 (0)  0      1024   0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M6|MEM_MOD:MM_ram|altdpram:altdpram_component
         |lpm_counter:w_addr_reg_rtl_877|    7 (0)  7      0      0      0      0 (0)  0
(0)    7 (0)  |project_megafunctions|MM:MM1|MM_RAM:M6|lpm_counter:w_addr_reg_rtl_877
            |alt_synch_counter:wysi_counter| 7 (7)  7      0      0      0      0 (0)  0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M6|lpm_counter:w_addr_reg_rtl_877|alt_synch_count
er:wysi_counter
      |MM_RAM:M7|     116 (109)      38     1024   0      0      78 (78)0 (0)  38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M7
         |MEM_MOD:MM_ram|     0 (0)  0      1024   0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M7|MEM_MOD:MM_ram
            |altdpram:altdpram_component|    0 (0)  0      1024   0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M7|MEM_MOD:MM_ram|altdpram:altdpram_component
         |lpm_counter:w_addr_reg_rtl_876|    7 (0)  7      0      0      0      0 (0)  0
(0)    7 (0)  |project_megafunctions|MM:MM1|MM_RAM:M7|lpm_counter:w_addr_reg_rtl_876
            |alt_synch_counter:wysi_counter| 7 (7)  7      0      0      0      0 (0)  0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M7|lpm_counter:w_addr_reg_rtl_876|alt_synch_count
er:wysi_counter
      |MM_RAM:M8|     130 (123)      38     1024   0      0      92 (92)0 (0)  38 (31)
       |project_megafunctions|MM:MM1|MM_RAM:M8
         |MEM_MOD:MM_ram|     0 (0)  0      1024   0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M8|MEM_MOD:MM_ram
            |altdpram:altdpram_component|    0 (0)  0      1024   0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|MM:MM1|MM_RAM:M8|MEM_MOD:MM_ram|altdpram:altdpram_component
         |lpm_counter:w_addr_reg_rtl_874|    7 (0)  7      0      0      0      0 (0)  0
(0)    7 (0)  |project_megafunctions|MM:MM1|MM_RAM:M8|lpm_counter:w_addr_reg_rtl_874
            |alt_synch_counter:wysi_counter| 7 (7)  7      0      0      0      0 (0)  0
(0)    7 (7)
       |project_megafunctions|MM:MM1|MM_RAM:M8|lpm_counter:w_addr_reg_rtl_874|alt_synch_count
er:wysi_counter
      |PAC_SWCounter:SWC|     11 (11)6      0      0      0      5 (5)  0 (0)  6 (6)
       |project_megafunctions|MM:MM1|PAC_SWCounter:SWC
   |OIN:OIN1|  400 (394)      30     0      0      0      370 (370)      0 (0)  30 (24)
       |project_megafunctions|OIN:OIN1
      |lpm_counter:counter_rtl_873|   6 (0)  6      0      0      0      0 (0)  0 (0)  6
(0)    |project_megafunctions|OIN:OIN1|lpm_counter:counter_rtl_873
         |alt_synch_counter:wysi_counter|    6 (6)  6      0      0      0      0 (0)  0
(0)    6 (6)
       |project_megafunctions|OIN:OIN1|lpm_counter:counter_rtl_873|alt_synch_counter:wysi_cou
nter
   |PAC_Full:PAC|     1801 (0)       438    32     0      0      1363 (0)       0 (0)  438
(0)    |project_megafunctions|PAC_Full:PAC
      |PAC_600:P_600|  355 (341)      80     0      0      0      275 (267)      0 (0)  80
(74)   |project_megafunctions|PAC_Full:PAC|PAC_600:P_600
         |PAC_SWCounter:SW_Count|     14 (14)6      0      0      0      8 (8)  0 (0)  6
(6)    |project_megafunctions|PAC_Full:PAC|PAC_600:P_600|PAC_SWCounter:SW_Count
      |PAC_650:P_650|  1225 (1211)    212    0      0      0      1013 (1005)    0 (0)  212
(206)  |project_megafunctions|PAC_Full:PAC|PAC_650:P_650
         |PAC_SWCounter:SW_Count_650| 14 (14)6      0      0      0      8 (8)  0 (0)  6
(6)    |project_megafunctions|PAC_Full:PAC|PAC_650:P_650|PAC_SWCounter:SW_Count_650
      |PAC_Header:Header1|    67 (3)  67     0      0      0      0 (0)  0 (0)  67 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header1
```

```
        |shift_reg2:shift1|  64 (0)  64     0      0      0      0 (0)  0 (0)  64 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header1|shift_reg2:shift1
           |lpm_shiftreg:lpm_shiftreg_component|    64 (64) 64     0      0      0      0
(0)    0 (0)  64 (64)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header1|shift_reg2:shift1|lpm_shiftreg:
lpm_shiftreg_component
      |PAC_Header:Header2|    6 (3)   6      0      0      0      0 (0)  0 (0)  6 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header2
         |shift_reg2:shift1|  3 (0)   3      0      0      0      0 (0)  0 (0)  3 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header2|shift_reg2:shift1
           |lpm_shiftreg:lpm_shiftreg_component|    3 (3)   3      0      0      0      0
(0)    0 (0)  3 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header2|shift_reg2:shift1|lpm_shiftreg:
lpm_shiftreg_component
      |PAC_Header:Header3|    6 (3)   6      0      0      0      0 (0)  0 (0)  6 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header3
         |shift_reg2:shift1|  3 (0)   3      0      0      0      0 (0)  0 (0)  3 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header3|shift_reg2:shift1
           |lpm_shiftreg:lpm_shiftreg_component|    3 (3)   3      0      0      0      0
(0)    0 (0)  3 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header3|shift_reg2:shift1|lpm_shiftreg:
lpm_shiftreg_component
      |PAC_Header:Header4|    6 (3)   6      0      0      0      0 (0)  0 (0)  6 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header4
         |shift_reg2:shift1|  3 (0)   3      0      0      0      0 (0)  0 (0)  3 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header4|shift_reg2:shift1
           |lpm_shiftreg:lpm_shiftreg_component|    3 (3)   3      0      0      0      0
(0)    0 (0)  3 (3)
       |project_megafunctions|PAC_Full:PAC|PAC_Header:Header4|shift_reg2:shift1|lpm_shiftreg:
lpm_shiftreg_component
      |PAC_Memory:Mem1|       26 (26) 13     8      0      0      13 (13) 0 (0)  13 (13)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem1
         |ram_pac:ram1|       0 (0)   0      8      0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem1|ram_pac:ram1
           |altdpram:altdpram_component|    0 (0)   0      8      0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem1|ram_pac:ram1|altdpram:altdpram_com
ponent
      |PAC_Memory:Mem2|       27 (26) 13     8      0      0      14 (13) 0 (0)  13 (13)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem2
         |ram_pac:ram1|       1 (0)   0      8      0      0      1 (0)  0 (0)  0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem2|ram_pac:ram1
           |altdpram:altdpram_component|    1 (1)   0      8      0      0      1 (1)  0
(0)    0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem2|ram_pac:ram1|altdpram:altdpram_com
ponent
      |PAC_Memory:Mem3|       26 (26) 13     8      0      0      13 (13) 0 (0)  13 (13)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem3
         |ram_pac:ram1|       0 (0)   0      8      0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem3|ram_pac:ram1
           |altdpram:altdpram_component|    0 (0)   0      8      0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem3|ram_pac:ram1|altdpram:altdpram_com
ponent
      |PAC_Memory:Mem4|       28 (28) 13     8      0      0      15 (15) 0 (0)  13 (13)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem4
         |ram_pac:ram1|       0 (0)   0      8      0      0      0 (0)  0 (0)  0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem4|ram_pac:ram1
           |altdpram:altdpram_component|    0 (0)   0      8      0      0      0 (0)  0
(0)    0 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_Memory:Mem4|ram_pac:ram1|altdpram:altdpram_com
ponent
      |PAC_NCFSM:NCFSM|       29 (20) 9      0      0      0      20 (20) 0 (0)  9 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_NCFSM:NCFSM
         |lpm_counter:counter_rtl_884|       9 (0)   9      0      0      0      0 (0)  0
(0)    9 (0)
       |project_megafunctions|PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884
           |alt_synch_counter:wysi_counter| 9 (9)   9      0      0      0      0 (0)  0
(0)    9 (9)
       |project_megafunctions|PAC_Full:PAC|PAC_NCFSM:NCFSM|lpm_counter:counter_rtl_884|alt_sy
nch_counter:wysi_counter
```

```
   |PS:PS1|    500 (500)      262    0       0       0       238 (238)      0 (0)   262 (262)
       |project_megafunctions|PS:PS1
```

```
/---------------------------------------/
PIN File
/---------------------------------------/
-- Copyright (C) 1991-2003 Altera Corporation
 -- Any  megafunction  design,  and related netlist (encrypted  or  decrypted),
 -- support information,  device programming or simulation file,  and any other
 -- associated  documentation or information  provided by  Altera  or a partner
 -- under  Altera's   Megafunction   Partnership   Program  may  be  used  only
 -- to program  PLD  devices (but not masked  PLD  devices) from  Altera.   Any
 -- other  use  of such  megafunction  design,  netlist,  support  information,
 -- device programming or simulation file,  or any other  related documentation
 -- or information  is prohibited  for  any  other purpose,  including, but not
 -- limited to  modification,  reverse engineering,  de-compiling, or use  with
 -- any other  silicon devices,  unless such use is  explicitly  licensed under
 -- a separate agreement with  Altera  or a megafunction partner.  Title to the
 -- intellectual property,  including patents,  copyrights,  trademarks,  trade
 -- secrets,  or maskworks,  embodied in any such megafunction design, netlist,
 -- support  information,  device programming or simulation file,  or any other
 -- related documentation or information provided by  Altera  or a megafunction
 -- partner, remains with Altera, the megafunction partner, or their respective
 -- licensors. No other licenses, including any licenses needed under any third
 -- party's intellectual property, are provided herein.
 -------------------------------------------------------------------------------



 -------------------------------------------------------------------------------
 -- NC         : No Connect. This pin has no internal connection to the device.
 -- VCC_INT    : Dedicated power pin, which MUST be connected to VCC  (1.8V).
 -- VCC_IO     : Dedicated power pin, which MUST be connected to VCC  (Refer to
 --              the table below for voltage).
 -- GND        : Dedicated ground pin, which MUST be connected to GND.
 -- GND+       : Unused input. This pin should be connected to GND. It may also
 --              be connected  to a  valid signal  on the board  (low, high, or
 --              toggling)  if that signal is required for a different revision
 --              of the design.
 -- GND*       : Unused  I/O  pin.   This pin can either be left unconnected or
 --              connected to GND.  Connecting this pin to GND will improve the
 --              device's immunity to noise.
 -------------------------------------------------------------------------------


Quartus II Version 3.0 Build 223 08/14/2003 Service Pack 1 SJ Full Version
CHIP  "Project_Megafunctions"  ASSIGNED TO AN: EP20K400EFC672-3

Pin Name/Usage               : Location  : Dir.   : I/O Standard      : Voltage : I/O Bank  :
User Assignment
-------------------------------------------------------------------------------------------------
----------------
GND                          : A2        : gnd    :                   :         :           :
VCC_INT                      : A3        : power  :                   : 1.8V    :           :
GND*                         : A4        :        :                   :         : 1         :
GND*                         : A5        :        :                   :         : 1         :
VCC_IO                       : A6        : power  :                   : 3.3V    :           :
GND*                         : A7        :        :                   :         : 1         :
GND                          : A8        : gnd    :                   :         :           :
GND*                         : A9        :        :                   :         : 1         :
GND*                         : A10       :        :                   :         : 1         :
GND*                         : A11       :        :                   :         : 1         :
GND*                         : A12       :        :                   :         : 1         :
VCC_IO                       : A13       : power  :                   : 3.3V    :           :
GND                          : A14       : gnd    :                   :         :           :
NC                           : A15       :        :                   :         :           :
NC                           : A16       :        :                   :         :           :
GND*                         : A17       :        :                   :         : 2         :
GND*                         : A18       :        :                   :         : 2         :
```

| | | | | | | |
|---|---|---|---|---|---|---|
| GND | A19 | gnd | | | | |
| GND* | A20 | | | | 2 | |
| VCC_IO | A21 | power | | 3.3V | | |
| GND* | A22 | | | | 2 | |
| GND* | A23 | | | | 2 | |
| VCC_INT | A24 | power | | 1.8V | | |
| GND | A25 | gnd | | | | |
| GND | B1 | gnd | | | | |
| GND | B2 | gnd | | | | |
| VCC_INT | B3 | power | | 1.8V | | |
| GND* | B4 | | | | 1 | |
| GND* | B5 | | | | 1 | |
| GND | B6 | gnd | | | | |
| GND* | B7 | | | | 1 | |
| VCC_INT | B8 | power | | 1.8V | | |
| GND* | B9 | | | | 1 | |
| GND* | B10 | | | | 1 | |
| GND* | B11 | | | | 1 | |
| GND* | B12 | | | | 1 | |
| NC | B13 | | | | | |
| NC | B14 | | | | | |
| NC | B15 | | | | | |
| NC | B16 | | | | | |
| GND* | B17 | | | | 2 | |
| GND* | B18 | | | | 2 | |
| VCC_INT | B19 | power | | 1.8V | | |
| GND* | B20 | | | | 2 | |
| GND | B21 | gnd | | | | |
| GND* | B22 | | | | 2 | |
| GND* | B23 | | | | 2 | |
| VCC_INT | B24 | power | | 1.8V | | |
| GND | B25 | gnd | | | | |
| GND | B26 | gnd | | | | |
| VCC_INT | C1 | power | | 1.8V | | |
| VCC_INT | C2 | power | | 1.8V | | |
| GND | C3 | gnd | | | | |
| GND* | C4 | | | | 8 | |
| GND* | C5 | | | | 1 | |
| GND* | C6 | | | | 1 | |
| GND* | C7 | | | | 1 | |
| GND* | C8 | | | | 1 | |
| GND* | C9 | | | | 1 | |
| GND* | C10 | | | | 1 | |
| NC | C11 | | | | | |
| NC | C12 | | | | | |
| GND | C13 | gnd | | | | |
| NC | C14 | | | | | |
| NC | C15 | | | | | |
| NC | C16 | | | | | |
| GND* | C17 | | | | 2 | |
| GND* | C18 | | | | 2 | |
| GND* | C19 | | | | 2 | |
| GND* | C20 | | | | 2 | |
| GND* | C21 | | | | 2 | |
| GND* | C22 | | | | 2 | |
| GND* | C23 | | | | 2 | |
| GND | C24 | gnd | | | | |
| VCC_INT | C25 | power | | 1.8V | | |
| VCC_INT | C26 | power | | 1.8V | | |
| GND* | D1 | | | | 7 | |
| GND* | D2 | | | | 7 | |
| VCC_INT | D3 | power | | 1.8V | | |
| GND | D4 | gnd | | | | |
| GND* | D5 | | | | 8 | |
| GND* | D6 | | | | 1 | |
| GND* | D7 | | | | 1 | |
| GND* | D8 | | | | 1 | |
| GND* | D9 | | | | 1 | |
| GND* | D10 | | | | 1 | |
| GND* | D11 | | | | 1 | |
| GND* | D12 | | | | 1 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | D13 | | | | 1 | |
| GND* | D14 | | | | 1 | |
| GND* | D15 | | | | 1 | |
| GND* | D16 | | | | 2 | |
| GND* | D17 | | | | 2 | |
| GND* | D18 | | | | 2 | |
| GND* | D19 | | | | 2 | |
| GND* | D20 | | | | 2 | |
| GND* | D21 | | | | 2 | |
| GND* | D22 | | | | 2 | |
| GND | D23 | gnd | | | | |
| VCC_INT | D24 | power | | 1.8V | | |
| GND* | D25 | | | | 3 | |
| GND* | D26 | | | | 3 | |
| GND* | E1 | | | | 7 | |
| GND* | E2 | | | | 7 | |
| GND* | E3 | | | | 8 | |
| GND* | E4 | | | | 8 | |
| GND* | E5 | | | | 8 | |
| GND* | E6 | | | | 1 | |
| GND* | E7 | | | | 1 | |
| GND* | E8 | | | | 1 | |
| GND* | E9 | | | | 1 | |
| GND* | E10 | | | | 1 | |
| GND* | E11 | | | | 1 | |
| GND* | E12 | | | | 1 | |
| GND* | E13 | | | | 2 | |
| GND* | E14 | | | | 2 | |
| GND* | E15 | | | | 2 | |
| GND* | E16 | | | | 2 | |
| GND* | E17 | | | | 2 | |
| GND* | E18 | | | | 2 | |
| GND* | E19 | | | | 2 | |
| GND* | E20 | | | | 2 | |
| GND* | E21 | | | | 2 | |
| GND* | E22 | | | | 3 | |
| GND* | E23 | | | | 2 | |
| GND* | E24 | | | | 2 | |
| GND* | E25 | | | | 3 | |
| GND* | E26 | | | | 3 | |
| GND* | F1 | | | | 7 | |
| GND* | F2 | | | | 7 | |
| GND* | F3 | | | | 8 | |
| GND* | F4 | | | | 8 | |
| GND* | F5 | | | | 8 | |
| GND* | F6 | | | | 1 | |
| GND* | F7 | | | | 1 | |
| GND* | F8 | | | | 1 | |
| GND* | F9 | | | | 1 | |
| GND* | F10 | | | | 1 | |
| GND* | F11 | | | | 1 | |
| GND* | F12 | | | | 1 | |
| GND+ | F13 | | | | 1 | |
| TRST | F14 | input | | | | |
| GND* | F15 | | | | 2 | |
| GND* | F16 | | | | 2 | |
| GND* | F17 | | | | 2 | |
| GND* | F18 | | | | 2 | |
| GND* | F19 | | | | 2 | |
| GND* | F20 | | | | 2 | |
| GND* | F21 | | | | 3 | |
| GND* | F22 | | | | 3 | |
| GND* | F23 | | | | 2 | |
| GND* | F24 | | | | 2 | |
| GND* | F25 | | | | 3 | |
| GND* | F26 | | | | 3 | |
| GND* | G1 | | | | 7 | |
| GND* | G2 | | | | 7 | |
| GND* | G3 | | | | 8 | |
| GND* | G4 | | | | 8 | |
| GND* | G5 | | | | 8 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | G6 | | | | 8 | |
| GND* | G7 | | | | 8 | |
| GND* | G8 | | | | 8 | |
| GND* | G9 | | | | 1 | |
| GND* | G10 | | | | 1 | |
| GND* | G11 | | | | 1 | |
| GND* | G12 | | | | 1 | |
| TDO | G13 | output | | | | |
| nCEO | G14 | output | | | | |
| GND* | G15 | | | | 2 | |
| GND* | G16 | | | | 2 | |
| GND* | G17 | | | | 2 | |
| GND* | G18 | | | | 2 | |
| GND* | G19 | | | | 2 | |
| GND* | G20 | | | | 3 | |
| GND* | G21 | | | | 3 | |
| GND* | G22 | | | | 3 | |
| GND* | G23 | | | | 2 | |
| GND* | G24 | | | | 3 | |
| GND* | G25 | | | | 3 | |
| GND* | G26 | | | | 3 | |
| GND* | H1 | | | | 7 | |
| GND* | H2 | | | | 7 | |
| GND* | H3 | | | | 8 | |
| GND* | H4 | | | | 8 | |
| GND* | H5 | | | | 8 | |
| GND* | H6 | | | | 8 | |
| GND* | H7 | | | | 8 | |
| GND | H8 | gnd | | | | |
| GND* | H9 | | | | 8 | |
| GND* | H10 | | | | 1 | |
| GND* | H11 | | | | 1 | |
| GND* | H12 | | | | 1 | |
| GND* | H13 | | | | 1 | |
| reset<br>N | H14 | input | LVTTL | | 1 | |
| GND* | H15 | | | | 2 | |
| GND* | H16 | | | | 2 | |
| GND* | H17 | | | | 2 | |
| GND* | H18 | | | | 2 | |
| GND | H19 | gnd | | | | |
| GND* | H20 | | | | 3 | |
| GND* | H21 | | | | 3 | |
| GND* | H22 | | | | 3 | |
| GND* | H23 | | | | 2 | |
| GND* | H24 | | | | 3 | |
| GND* | H25 | | | | 3 | |
| GND* | H26 | | | | 3 | |
| GND* | J1 | | | | 7 | |
| GND* | J2 | | | | 7 | |
| GND* | J3 | | | | 8 | |
| GND* | J4 | | | | 8 | |
| GND* | J5 | | | | 8 | |
| GND* | J6 | | | | 8 | |
| PS1_port3_out<br>N | J7 | output | LVTTL | | 8 | |
| GND* | J8 | | | | 8 | |
| GND | J9 | gnd | | | | |
| VCC_IO | J10 | power | | 3.3V | | |
| GND* | J11 | | | | 1 | |
| GND* | J12 | | | | 1 | |
| GND* | J13 | | | | 1 | |
| GND* | J14 | | | | 1 | |
| GND* | J15 | | | | 1 | |
| GND* | J16 | | | | 2 | |
| GND* | J17 | | | | 2 | |
| GND | J18 | gnd | | | | |
| GND* | J19 | | | | 2 | |
| GND* | J20 | | | | 3 | |
| GND* | J21 | | | | 3 | |
| GND* | J22 | | | | 3 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | J23 | | | | 3 | |
| GND* | J24 | | | | 3 | |
| GND* | J25 | | | | 3 | |
| GND* | J26 | | | | 3 | |
| GND* | K1 | | | | 7 | |
| GND* | K2 | | | | 7 | |
| GND* | K3 | | | | 8 | |
| GND* | K4 | | | | 8 | |
| PS1_port2_out<br>N | K5 | output | LVTTL | | 8 | |
| GND* | K6 | | | | 8 | |
| GND* | K7 | | | | 8 | |
| GND* | K8 | | | | 8 | |
| VCC_IO | K9 | power | | 3.3V | | |
| GND | K10 | gnd | | | | |
| VCC_INT | K11 | power | | 1.8V | | |
| GND* | K12 | | | | 1 | |
| GND* | K13 | | | | 1 | |
| GND* | K14 | | | | 1 | |
| GND* | K15 | | | | 2 | |
| VCC_IO | K16 | power | | 3.3V | | |
| GND | K17 | gnd | | | | |
| GND* | K18 | | | | 2 | |
| GND* | K19 | | | | 3 | |
| GND* | K20 | | | | 3 | |
| GND* | K21 | | | | 3 | |
| GND* | K22 | | | | 3 | |
| GND* | K23 | | | | 3 | |
| GND* | K24 | | | | 3 | |
| GND* | K25 | | | | 3 | |
| GND* | K26 | | | | 3 | |
| GND* | L1 | | | | 7 | |
| GND* | L2 | | | | 7 | |
| GND* | L3 | | | | 8 | |
| GND* | L4 | | | | 8 | |
| GND* | L5 | | | | 8 | |
| GND* | L6 | | | | 8 | |
| PS1_port1_out<br>N | L7 | output | LVTTL | | 8 | |
| GND* | L8 | | | | 8 | |
| GND* | L9 | | | | 8 | |
| VCC_INT | L10 | power | | 1.8V | | |
| GND | L11 | gnd | | | | |
| VCC_IO | L12 | power | | 3.3V | | |
| GND | L13 | gnd | | | | |
| GND* | L14 | | | | 2 | |
| VCC_INT | L15 | power | | 1.8V | | |
| GND | L16 | gnd | | | | |
| VCC_IO | L17 | power | | 3.3V | | |
| GND* | L18 | | | | 3 | |
| GND* | L19 | | | | 3 | |
| GND* | L20 | | | | 4 | |
| GND* | L21 | | | | 4 | |
| GND* | L22 | | | | 4 | |
| GND* | L23 | | | | 4 | |
| GND* | L24 | | | | 4 | |
| GND* | L25 | | | | 3 | |
| GND* | L26 | | | | 3 | |
| GND* | M1 | | | | 7 | |
| GND* | M2 | | | | 7 | |
| GND* | M3 | | | | 8 | |
| GND* | M4 | | | | 8 | |
| GND* | M5 | | | | 8 | |
| GND* | M6 | | | | 8 | |
| GND* | M7 | | | | 8 | |
| GND* | M8 | | | | 8 | |
| GND* | M9 | | | | 8 | |
| GND* | M10 | | | | 8 | |
| VCC_IO | M11 | power | | 3.3V | | |
| GND | M12 | gnd | | | | |
| VCC_INT | M13 | power | | 1.8V | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| VCC_IO | M14 | power | | 3.3V | | |
| GND | M15 | gnd | | | | |
| VCC_INT | M16 | power | | 1.8V | | |
| GND* | M17 | | | | 4 | |
| GND* | M18 | | | | 4 | |
| GND+ | M19 | | | | 4 | |
| GND* | M20 | | | | 4 | |
| GND* | M21 | | | | 4 | |
| GND* | M22 | | | | 4 | |
| GND* | M23 | | | | 4 | |
| GND* | M24 | | | | 4 | |
| GND* | M25 | | | | 3 | |
| GND* | M26 | | | | 3 | |
| GND | N1 | gnd | | | | |
| VCC_INT | N2 | power | | 1.8V | | |
| VCC_IO | N3 | power | | 3.3V | | |
| GND | N4 | gnd | | | | |
| PS1_port4_out<br>N | N5 | output | LVTTL | | 8 | |
| DATA0 | N6 | input | | | | |
| DCLK | N7 | bidir | | | | |
| GND+ | N8 | | | | 8 | |
| GND* | N9 | | | | 8 | |
| GND* | N10 | | | | 8 | |
| VCC_CKLK2 | N11 | power | | 1.8V | | |
| VCC_INT | N12 | power | | 1.8V | | |
| GND | N13 | gnd | | | | |
| GND | N14 | gnd | | | | |
| VCC_IO | N15 | power | | 3.3V | | |
| GND* | N16 | | | | 4 | |
| GND* | N17 | | | | 4 | |
| GND* | N18 | | | | 4 | |
| GND* | N19 | | | | 4 | |
| MSEL1 | N20 | input | | | | |
| MSEL0 | N21 | input | | | | |
| GND* | N22 | | | | 4 | |
| GND* | N23 | | | | 4 | |
| VCC_IO | N24 | power | | 3.3V | | |
| GND_CKLK3 | N25 | gnd | | | | |
| GND | N26 | gnd | | | | |
| GND | P1 | gnd | | | | |
| GND | P2 | gnd | | | | |
| GND | P3 | gnd | | | | |
| GND* | P4 | | | | 8 | |
| GND* | P5 | | | | 8 | |
| nCE | P6 | input | | | | |
| TDI | P7 | input | | | | |
| GND* | P8 | | | | 8 | |
| GND* | P9 | | | | 8 | |
| VCC_CKLK4 | P10 | power | | 1.8V | | |
| GND_CKLK2 | P11 | gnd | | | | |
| VCC_IO | P12 | power | | 3.3V | | |
| GND | P13 | gnd | | | | |
| GND | P14 | gnd | | | | |
| VCC_INT | P15 | power | | 1.8V | | |
| VCC_INT | P16 | | | | | |
| GND* | P17 | | | | 4 | |
| GND* | P18 | | | | 4 | |
| GND* | P19 | | | | 4 | |
| GND+ | P20 | | | | 4 | |
| NCONFIG | P21 | input | | | | |
| GND* | P22 | | | | 4 | |
| GND | P23 | gnd | | | | |
| VCC_INT | P24 | power | | 1.8V | | |
| VCC_INT | P25 | power | | 1.8V | | |
| GND | P26 | gnd | | | | |
| GND* | R1 | | | | 7 | |
| GND* | R2 | | | | 7 | |
| GND* | R3 | | | | 8 | |
| GND* | R4 | | | | 8 | |
| GND* | R5 | | | | 7 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| clock<br>N | R6 | input | LVTTL | | 8 | |
| GND* | R7 | | | | 8 | |
| GND* | R8 | | | | 7 | |
| GND* | R9 | | | | 8 | |
| GND_CKLK4 | R10 | gnd | | | | |
| VCC_INT | R11 | power | | 1.8V | | |
| GND | R12 | gnd | | | | |
| VCC_IO | R13 | power | | 3.3V | | |
| VCC_INT | R14 | power | | 1.8V | | |
| GND | R15 | gnd | | | | |
| VCC_IO | R16 | power | | 3.3V | | |
| GND* | R17 | | | | 4 | |
| GND* | R18 | | | | 4 | |
| GND* | R19 | | | | 4 | |
| GND* | R20 | | | | 4 | |
| GND* | R21 | | | | 4 | |
| GND* | R22 | | | | 4 | |
| GND* | R23 | | | | 4 | |
| GND* | R24 | | | | 4 | |
| GND* | R25 | | | | 3 | |
| GND* | R26 | | | | 3 | |
| GND* | T1 | | | | 7 | |
| GND* | T2 | | | | 7 | |
| GND* | T3 | | | | 7 | |
| GND* | T4 | | | | 7 | |
| GND* | T5 | | | | 7 | |
| GND* | T6 | | | | 8 | |
| GND* | T7 | | | | | |
| GND* | T8 | | | | 8 | |
| GND* | T9 | | | | 6 | |
| VCC_IO | T10 | power | | 3.3V | | |
| GND | T11 | gnd | | | | |
| VCC_INT | T12 | power | | 1.8V | | |
| GND* | T13 | | | | 6 | |
| GND* | T14 | | | | 5 | |
| VCC_IO | T15 | power | | 3.3V | | |
| GND | T16 | gnd | | | | |
| VCC_INT | T17 | power | | 1.8V | | |
| GND* | T18 | | | | 4 | |
| GND* | T19 | | | | 4 | |
| GND* | T20 | | | | 4 | |
| GND* | T21 | | | | 4 | |
| GND* | T22 | | | | 4 | |
| GND* | T23 | | | | 4 | |
| GND* | T24 | | | | 4 | |
| GND* | T25 | | | | 3 | |
| GND* | T26 | | | | 3 | |
| GND* | U1 | | | | 7 | |
| GND* | U2 | | | | 7 | |
| GND* | U3 | | | | 7 | |
| GND* | U4 | | | | 7 | |
| GND* | U5 | | | | 7 | |
| GND* | U6 | | | | 7 | |
| GND* | U7 | | | | 7 | |
| GND+ | U8 | | | | | |
| VCC_INT | U9 | power | | 1.8V | | |
| GND | U10 | gnd | | | | |
| VCC_IO | U11 | power | | 3.3V | | |
| GND* | U12 | | | | 6 | |
| GND* | U13 | | | | 6 | |
| GND* | U14 | | | | 5 | |
| GND* | U15 | | | | 5 | |
| VCC_INT | U16 | power | | 1.8V | | |
| GND | U17 | gnd | | | | |
| VCC_IO | U18 | power | | 3.3V | | |
| GND* | U19 | | | | 4 | |
| GND* | U20 | | | | 4 | |
| GND* | U21 | | | | 4 | |
| GND* | U22 | | | | 4 | |
| GND* | U23 | | | | 4 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | U24 | | | | 4 | |
| GND* | U25 | | | | 3 | |
| GND* | U26 | | | | 3 | |
| GND* | V1 | | | | 7 | |
| GND* | V2 | | | | 7 | |
| GND* | V3 | | | | 7 | |
| GND* | V4 | | | | 7 | |
| GND* | V5 | | | | 7 | |
| GND_CKOUT2 | V6 | gnd | | | | |
| VCC_CKOUT2 | V7 | power | | 3.3V | | |
| GND* | V8 | | | | 6 | |
| GND | V9 | gnd | | | | |
| VCC_IO | V10 | power | | 3.3V | | |
| GND* | V11 | | | | 6 | |
| GND* | V12 | | | | 6 | |
| GND* | V13 | | | | 6 | |
| GND* | V14 | | | | 5 | |
| GND* | V15 | | | | 5 | |
| GND* | V16 | | | | 5 | |
| VCC_IO | V17 | power | | 3.3V | | |
| GND | V18 | gnd | | | | |
| GND* | V19 | | | | 4 | |
| GND* | V20 | | | | 4 | |
| GND* | V21 | | | | 4 | |
| GND* | V22 | | | | 4 | |
| GND* | V23 | | | | 4 | |
| GND* | V24 | | | | 4 | |
| GND* | V25 | | | | 3 | |
| GND* | V26 | | | | 3 | |
| GND* | W1 | | | | 7 | |
| GND* | W2 | | | | 7 | |
| GND* | W3 | | | | 7 | |
| GND* | W4 | | | | 7 | |
| GND* | W5 | | | | 7 | |
| GND* | W6 | | | | 7 | |
| GND* | W7 | | | | 7 | |
| GND | W8 | gnd | | | | |
| GND* | W9 | | | | 6 | |
| GND* | W10 | | | | 6 | |
| GND* | W11 | | | | 6 | |
| GND* | W12 | | | | 6 | |
| GND* | W13 | | | | 6 | |
| GND* | W14 | | | | 6 | |
| GND* | W15 | | | | 5 | |
| GND* | W16 | | | | 5 | |
| GND* | W17 | | | | 5 | |
| GND* | W18 | | | | 5 | |
| GND | W19 | gnd | | | | |
| GND* | W20 | | | | 5 | |
| GND* | W21 | | | | 4 | |
| GND* | W22 | | | | 4 | |
| GND* | W23 | | | | 4 | |
| GND* | W24 | | | | 4 | |
| GND* | W25 | | | | 3 | |
| GND* | W26 | | | | 3 | |
| GND* | Y1 | | | | 7 | |
| GND* | Y2 | | | | 7 | |
| GND* | Y3 | | | | 6 | |
| GND* | Y4 | | | | 6 | |
| GND* | Y5 | | | | 7 | |
| GND* | Y6 | | | | 7 | |
| GND* | Y7 | | | | 6 | |
| GND* | Y8 | | | | 6 | |
| GND* | Y9 | | | | 6 | |
| GND* | Y10 | | | | 6 | |
| GND* | Y11 | | | | 6 | |
| GND* | Y12 | | | | 6 | |
| GND+ | Y13 | | | | 5 | |
| GND+ | Y14 | | | | 5 | |
| GND* | Y15 | | | | 5 | |
| GND* | Y16 | | | | 5 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | Y17 | | | | 5 | |
| GND* | Y18 | | | | 5 | |
| GND* | Y19 | | | | 5 | |
| GND* | Y20 | | | | 5 | |
| GND* | Y21 | | | | 4 | |
| ser_in[0] N | Y22 | input | LVTTL | | 4 | |
| GND* | Y23 | | | | 5 | |
| GND* | Y24 | | | | 5 | |
| GND* | Y25 | | | | 3 | |
| GND* | Y26 | | | | 3 | |
| GND* | AA1 | | | | 7 | |
| GND* | AA2 | | | | 7 | |
| GND* | AA3 | | | | 6 | |
| GND* | AA4 | | | | 6 | |
| GND* | AA5 | | | | 7 | |
| GND* | AA6 | | | | 7 | |
| GND* | AA7 | | | | 7 | |
| GND* | AA8 | | | | 6 | |
| GND* | AA9 | | | | 6 | |
| GND* | AA10 | | | | 6 | |
| ser_in[3] N | AA11 | input | LVTTL | | 6 | |
| CONF_DONE | AA12 | bidir | | | | |
| NSTATUS | AA13 | bidir | | | | |
| TCK | AA14 | input | | | | |
| TMS | AA15 | input | | | | |
| GND* | AA16 | | | | 5 | |
| GND* | AA17 | | | | 5 | |
| GND* | AA18 | | | | 5 | |
| GND* | AA19 | | | | 5 | |
| GND* | AA20 | | | | 5 | |
| GND* | AA21 | | | | 4 | |
| GND* | AA22 | | | | 4 | |
| GND* | AA23 | | | | 5 | |
| GND* | AA24 | | | | 5 | |
| GND* | AA25 | | | | 3 | |
| GND* | AA26 | | | | 3 | |
| GND* | AB1 | | | | 7 | |
| GND* | AB2 | | | | 7 | |
| GND* | AB3 | | | | 6 | |
| GND* | AB4 | | | | 6 | |
| GND* | AB5 | | | | 7 | |
| GND* | AB6 | | | | 7 | |
| GND* | AB7 | | | | 6 | |
| GND* | AB8 | | | | 6 | |
| GND* | AB9 | | | | 6 | |
| GND* | AB10 | | | | 6 | |
| GND* | AB11 | | | | 6 | |
| GND* | AB12 | | | | 6 | |
| GND* | AB13 | | | | 5 | |
| GND* | AB14 | | | | 5 | |
| GND* | AB15 | | | | 5 | |
| GND* | AB16 | | | | 5 | |
| GND* | AB17 | | | | 5 | |
| GND* | AB18 | | | | 5 | |
| GND* | AB19 | | | | 5 | |
| GND* | AB20 | | | | 5 | |
| ser_in[1] N | AB21 | input | LVTTL | | 4 | |
| GND* | AB22 | | | | 4 | |
| GND* | AB23 | | | | 5 | |
| GND* | AB24 | | | | 5 | |
| GND* | AB25 | | | | 3 | |
| GND* | AB26 | | | | 3 | |
| GND* | AC1 | | | | 7 | |
| GND* | AC2 | | | | 7 | |
| VCC_INT | AC3 | power | | 1.8V | | |
| GND | AC4 | gnd | | | | |
| GND* | AC5 | | | | 6 | |
| GND* | AC6 | | | | 6 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| GND* | AC7 | | | | 6 | |
| GND* | AC8 | | | | 6 | |
| GND* | AC9 | | | | 6 | |
| GND* | AC10 | | | | 6 | |
| GND* | AC11 | | | | 5 | |
| GND* | AC12 | | | | 5 | |
| GND* | AC13 | | | | 5 | |
| GND* | AC14 | | | | 5 | |
| GND* | AC15 | | | | 5 | |
| GND* | AC16 | | | | 5 | |
| ser_in[2] | AC17 | input | LVTTL | | 5 | |
| N | | | | | | |
| GND* | AC18 | | | | 5 | |
| GND* | AC19 | | | | 5 | |
| GND* | AC20 | | | | 5 | |
| GND* | AC21 | | | | 5 | |
| GND* | AC22 | | | | 5 | |
| GND | AC23 | gnd | | | | |
| VCC_INT | AC24 | power | | 1.8V | | |
| GND_CKLK3 | AC25 | gnd | | | | |
| VCC_CKLK3 | AC26 | power | | 1.8V | | |
| VCC_INT | AD1 | power | | 1.8V | | |
| VCC_INT | AD2 | power | | 1.8V | | |
| GND | AD3 | gnd | | | | |
| GND* | AD4 | | | | 6 | |
| GND* | AD5 | | | | 6 | |
| GND* | AD6 | | | | 6 | |
| GND* | AD7 | | | | 6 | |
| GND* | AD8 | | | | 6 | |
| GND* | AD9 | | | | 6 | |
| GND* | AD10 | | | | 5 | |
| NC | AD11 | | | | | |
| NC | AD12 | | | | | |
| GND | AD13 | gnd | | | | |
| NC | AD14 | | | | | |
| NC | AD15 | | | | | |
| NC | AD16 | | | | | |
| GND* | AD17 | | | | 5 | |
| GND* | AD18 | | | | 5 | |
| GND* | AD19 | | | | 5 | |
| GND* | AD20 | | | | 5 | |
| GND* | AD21 | | | | 5 | |
| GND* | AD22 | | | | 5 | |
| GND* | AD23 | | | | 5 | |
| GND | AD24 | gnd | | | | |
| VCC_INT | AD25 | power | | 1.8V | | |
| VCC_INT | AD26 | power | | 1.8V | | |
| GND | AE1 | gnd | | | | |
| GND | AE2 | gnd | | | | |
| VCC_INT | AE3 | power | | 1.8V | | |
| GND* | AE4 | | | | 6 | |
| GND* | AE5 | | | | 6 | |
| GND | AE6 | gnd | | | | |
| GND* | AE7 | | | | 6 | |
| VCC_INT | AE8 | power | | 1.8V | | |
| GND* | AE9 | | | | 6 | |
| GND* | AE10 | | | | 6 | |
| GND* | AE11 | | | | 6 | |
| NC | AE12 | | | | | |
| NC | AE13 | | | | | |
| NC | AE14 | | | | | |
| NC | AE15 | | | | | |
| GND* | AE16 | | | | 5 | |
| GND* | AE17 | | | | 5 | |
| GND_CKLK1 | AE18 | gnd | | | | |
| VCC_INT | AE19 | power | | 1.8V | | |
| GND+ | AE20 | | | | | |
| GND | AE21 | gnd | | | | |
| GND_CKOUT1 | AE22 | gnd | | | | |
| GND* | AE23 | | | | | |
| VCC_INT | AE24 | power | | 1.8V | | |

```
GND                          : AE25      : gnd    :                  :         :          :
GND                          : AE26      : gnd    :                  :         :          :
GND                          : AF2       : gnd    :                  :         :          :
VCC_INT                      : AF3       : power  :                  : 1.8V    :          :
GND*                         : AF4       :        :                  :         : 6        :
GND*                         : AF5       :        :                  :         : 6        :
VCC_IO                       : AF6       : power  :                  : 3.3V    :          :
GND*                         : AF7       :        :                  :         : 6        :
GND                          : AF8       : gnd    :                  :         :          :
GND*                         : AF9       :        :                  :         : 6        :
GND*                         : AF10      :        :                  :         : 6        :
GND*                         : AF11      :        :                  :         : 6        :
NC                           : AF12      :        :                  :         :          :
VCC_IO                       : AF13      : power  :                  : 3.3V    :          :
GND                          : AF14      : gnd    :                  :         :          :
NC                           : AF15      :        :                  :         :          :
GND*                         : AF16      :        :                  :         : 5        :
GND*                         : AF17      :        :                  :         : 5        :
VCC_CKLK1                    : AF18      : power  :                  : 1.8V    :          :
GND                          : AF19      : gnd    :                  :         :          :
GND*                         : AF20      :        :                  :         : 4        :
VCC_IO                       : AF21      : power  :                  : 3.3V    :          :
VCC_CKOUT1                   : AF22      : power  :                  : 3.3V    :          :
GND*                         : AF23      :        :                  :         : 3        :
VCC_INT                      : AF24      : power  :                  : 1.8V    :          :
GND                          : AF25      : gnd    :                  :         :          :
:
```

---

## Matlab Code

---

```
%test_sw.m
%(c) 2004 B.R. Phelps, North Carolina State University
%Created by: Brian R. Phelps    12/22/03
%Project: Sliding Window Packet Switch FPGA Implementation
%                           from Dr. Sanjeev Kumar's packet switch algorithm.
%Module Description:
% test_pac.m
% This is the Parent Routine for testing the SW algorithm

% This file uses the following PAC algorithms:
%   1. PAC() Parameter assignment circuit
%   2. IIN() Input interconnection network
%   3. WO()  Write operation
%   4. RO()  Read operation
%   5. OIN() Output interconnection network
% The following functions display the inner workings of the algorithm:
%   1. write_file() write output to a file
%   2. decode() grab Param's from packet
%   3  encode() put Param's into packet
clear; close all;
tic;
Update_Variables=-1;    % Packets from b_traffic will never have this value
                    %   so we will use it as a signal to update the counters
%-------Parameter declarations----------------------------
global m sigma NxN p;   %Many Functions need to use these parameters
```

```
global mem_array OSA    % These are declared global b/c they are shared b/w
                    %   the read (RO) & write (WO) & write_file operations's

m=8;        % Number of Memory modules
sigma=12;   % Number of OSV
NxN=4;      % Number of Inputs
p=2;        % Number of Scan planes

Eb=9;%9;   % Burstiness - Traffic Parameters
L=.9999;  % Load - Traffic Parameters
max_cycles=250000;% Number of Traffic cycles - Traffic Parameters

flood_port=0; % Set to output port # to flood with packets.
            %     0 means b_traffic is used to generate packets
%-----------------Generate test traffic--------------------

if flood_port
   [traffic]=flood_port*ones(max_cycles,NxN);
else
   [traffic] = b_traffic(Eb, L, NxN, max_cycles);
end
%traffic=[1 1 1 1; 2 2 2 2]; % For debugging verilog

[L,W]=size(traffic);

[traffic]=[traffic; zeros(sigma*2+4,NxN)];  % Pad with zeros to clear pipeline
[L,W]=size(traffic);

%------------Intialization of memory & circuitry--------------------------
PAC(0,0); % reset the PAC to Fig4 initial values
RO(0,0.0);  % (I only reset functions that need local counters)
packet_X=zeros(NxN,1);      % Clear output of PAC
delay=zeros(NxN,1);         %  Not part of SW algorithm
mem_array=zeros(m,sigma);   % initialize the memory to 0 (could be done in the reset of WO();)
mem_port_c1=zeros(m,1);  mem_port_c2=zeros(m,1); %initialize to 0's for pipelining
oini=zeros(m,1);            %initialize to 0's for pipelining
OSA=zeros(m,sigma);         % initialize the Output Scan Array to 0's

write_file(1,0);  % open the Output file for data
%---------------SW Switch cycle loop----------------------------
for set_index=1:L     % Cycle through each set of packets

   oino=OIN(oini);               % Perform Output Interconnect Algorithm & put output in oini
length(oini)=NxN
   oini=RO(1,1);                 % Perform Read Operation & put output in oini length(oini)=m
   WO(mem_port_c1, mem_port_c2);   % Write Operation Algorithm (outputs to mem_array)

   % ------ Code Not part of SW algorithm -------------------------------

   write_file(2,delay,oino);         % Display the memory to a viewable "cycle by cycle" file (from data in
mem_array)
   delay=packet_X;             % also displays the output of the PAC that is being written to the
mem_array in that cycle
   % ------ End Code Not part of SW algorithm -------------------------------
```

```
   [mem_port_c1, mem_port_c2]=IIN(packet_X); % Perform Input Interconnect Algorithm & put output
in mem_port_c1, mem_port_c2
                                % mem_port_c1 is the output of the IIN for cycle 1 for the Memory
Modules

   % The port read loop is executed 1 time each switch cycle
   %   Cycles throught the input ports
   for port=1:NxN
      packet_X(port)=PAC(traffic(set_index,port),1);     % packet_X array will have the format:
encode([i,j,k,d])
   end

   RO(1,Update_Variables);   % Update Variables (Increment RO.Switch Cycle,SW_j,SW_k)
   PAC(Update_Variables,1);  % Update Variables (Increment PAC.Switch Cycle, etc)

end
%---------------End of SW Switch cycle loop----------------------------
%traffic
write_file(3,0,0);  % Close the viewable cycle by cycle file;
toc
clear;

system('zwrite brphelps -m Matlab Simulation done');
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function [ traffic ] = b_traffic(Eb, L, NxN, max_cycles)
%This function generates bursty traffic using the two-state ON-OFF model
%Eb          => Average Burst Size
%L           => Load
%NxN         => Number of Ports
%max_cycles   => Maximum Number of Cycles;


p = 1/Eb;        %Characterizes the duration of the active period
r = L*p/(L*p-L+1);      %Characterizes the duration of the idle period
traffic = zeros(max_cycles,NxN);     %Initializes the matrix to zeros



total_packets = 0;
for j = 1:NxN,    %Executes program for each port

   cycle = 1;
   status = 0;   %The initial period is idle
   train = 0;
   packet_number = 0;


   while 1,
```

```
if status == 1,   %ACTIVE MODE

  port = ceil(NxN*rand(1));   %One port is selected in a random fashion

  traffic(cycle,j) = port;   %At least one packet is sent
  packet_number = packet_number + 1;
  cycle = cycle + 1;

  if cycle > max_cycles,     %%%Break if we have the
    train = train +1;       %%%maximum number of cycles
    break;
  end


   while 1,
       if rand(1) > p         %Stay in active mode
         traffic(cycle,j) = port;
         packet_number = packet_number + 1;
         cycle = cycle + 1;

       else                  %Change to idle mode
         status = 0;
         train = train +1;
         break;

       end

       if cycle > max_cycles,      %%%Break if we have the
         train = train +1;        %%%maximum number of cycles
         break;
       end
   end


elseif status == 0,   %IDLE MODE

   while 1,
       if rand(1) > r   %Stay in idle mode
         %traffic(cycle,j) = 0;
         cycle = cycle + 1;

       else          %Change to active mode
         status = 1;
         break;

       end

       if cycle > max_cycles,   %%%Break if we have the
         break;               %%%maximum number of cycles
       end

   end


end
```

```
      if cycle > max_cycles,   %%%Break if we have the
        break;               %%%maximum number of cycles
      end


   end

   burst_size(j) = packet_number/train;

   load(j) = packet_number/cycle;

   total_packets = total_packets + packet_number;

end



burst_size = sum(burst_size)/max(size(burst_size));

load = sum(load)/max(size(load));
load = total_packets/(NxN*max_cycles);
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function [id,jd,kd,dd]=decode(packet)
%(c) 2004 B.R. Phelps, North Carolina State University
% Decodes id,jd,kd,dd from there respective locations in packet array
%   Assumes they are stored in 8 bit slots in the order above
%   id,jd,kd,dd come back as a column array

if ((packet==0)&(length(packet)==1))% This is not neccessary & is done to speed things up
   length(packet);
   id=0;jd=0;kd=0;dd=0;    % This is not neccessary & is done to speed things up
   return
end


for k1=1:length(packet)
   dat=packet(k1);     % extract data from the slot
   id(k1,1)=floor(mod(dat,2^8)/2^0);     % mask the bits for id
   jd(k1,1)=floor(mod(dat,2^16)/2^8);    % mask the bits for jd
   kd(k1,1)=floor(mod(dat,2^24)/2^16);   % mask the bits for kd
   dd(k1,1)=floor(mod(dat,2^32)/2^24);   % mask the bits for dd
end

return
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function packet=encode(dat)
% Encodes id,jd,kd,dd from there respective locations in array dat
%   Assumes they are stored in 8 bit slots in the order above
%   packet value is returned
%   Basically makes 4 smaller numbers into 1 big number

id=dat(:,1);jd=dat(:,2);kd=dat(:,3);dd=dat(:,4);

packet=id+jd*(2^8)+kd*(2^16)+dd*(2^24);

packet=packet';
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function [out_port_c1, out_port_c2]=IIN(in_port)
global m sigma NxN p

% Input interconnection algorithm
% Works regardless of NxN & m
%   however it simulates the hardware algorithm if (NxN:m)=(4:8)
% Simulates a 2 cycle input interconnection algorithm (to handle special case of double identical id
values)
% Assumes packets come in (on each port) as : encode([id, jd, kd, d]); where each port is a row in the
array
%
%   Example function output:
% [out_port_c1, out_port_c2]=input_intcn_netwk(4,8,encode([8 1 3 1; 2 2 3 2; 5 4 3 2; 8 3 3 2;]))
% out_port_c1 =
%     0    0    0    0
%     2    2    3    2
%     0    0    0    0
%     0    0    0    0
%     5    4    3    2
%     0    0    0    0
%     0    0    0    0
```

```
%     8    1    3    1
% out_port_c2 =
%
%     0    0    0    0
%     0    0    0    0
%     0    0    0    0
%     0    0    0    0
%     0    0    0    0
%     0    0    0    0
%     0    0    0    0
%     8    3    3    2

[id,jd,kd,dd]=decode(in_port);  % Extract parameters into column vectors
packet(:,1:4)=[id jd kd dd];    % store them into packet array

out_port_c1=zeros(m,1);  out_port_c2=zeros(m,1);  % Set output ports to an empty array (m rows
NxN columns)

% Cycle through the input ports
for k1= 1:NxN
   if (packet(k1,1))
   if ( out_port_c1(packet(k1,1))==0 )              % is this id identical to another?
      out_port_c1(packet(k1,1))=(in_port(k1));      % Set the output port cycle 1 #id = the packet
   else
      out_port_c2(packet(k1,1))=(in_port(k1));      % Set the output port cycle 2 #id = the packet
   end
   end
end
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function oino=OIN(oini);
% Ouput interconnection algorithm
%   Input: oini - Ouput Interconnect Network Input
%   Output: oino- Ouput Interconnect Network Output
%
%   Globals: -Uses global parameters m, NxN
%
% Works regardless of NxN & m values
%   Assumes packets come in (on each oini) as : encode([id, jd, kd, d]);
%    where each OIN input is a row in the oini array
%   each row is one packet long
global NxN m p

oino=zeros(NxN,1);

% cycle through the Ouput Interconnect inputs
for k1=1:m

   if(oini(k1)>0)
      [id,jd,kd,dd]=decode(oini(k1)); % decode the packet
```

```
      oino(dd)=oini(k1);  % Output the given packet onto dd
   end
end
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University

function X=PAC(traffic,reset);
% PAC- parameter assignment circuit algorithm
% assigns X containing id jd kd & d
global m sigma NxN  p fid4 txt4  % We need the parameters to perform the algorithms

% The following tells Matlab not to erase these values after each function call.
% This simulates how hardware uses DFF's, it requires a reset to initialize.
persistent SW_j SW_k LC_j LC_k Qd ST temp
persistent switch_cycle

id=1; jd=1; kd=1;  %Not neccessary; helps avoid pesky warnings from Matlab

if ~reset % Active low reset to known state
   temp(1:m)=0; % Reset temp array for Dr. Kumar's id assignment algorithm
   switch_cycle=0; SW_j=1; SW_k=1; %Step 400 & 402
   LC_j=0; LC_k=0; Qd=zeros(1,NxN); ST=zeros(m,sigma);
```

```
   switch_cycle=switch_cycle+1; %Update counters   (Does step 404 for 1st time, to stay true to
algorithm)
   X=[0];
            fid4=fopen('verilog_port.dat','w');  % Open the file for verilog data
   return  % leave function now & Execute Steps 406-408
end

%-----doesn't make it here if reset is low---------------------------------
% else not_reset & keep going
if (traffic==-1) % Counter update signal, do nothing else
   temp(1:m)=0;
   % Remove this scan_plane's values(k) from ST (Step 408)
   %ST  % Debug ST before clearing
   for i_tst=1:m
      if(ST(i_tst,SW_j)==SW_k)
         ST(i_tst,SW_j)=0;
      end
   end

   % Remove a packet from each Qd(d) (Step 408)
   for d_tst=1:NxN
      if(Qd(d_tst)>0)
         Qd(d_tst)=Qd(d_tst)-1;
      end
   end
   %ST % Debug ST after clearing value

   %switch_cycle % display current switch cycle for plot
   %------Begin Counter update (Fig 4)---------------
   SW_j=mod(switch_cycle, sigma)+1; % Step 410

   if(SW_j==1)                % Step 412
      SW_k=mod(SW_k, p)+1;     % Step 414
   end
   switch_cycle=switch_cycle+1; %Update counter (Step 404)
   return  % Leave function now & perform 406-408
end

txt4=[sprintf('%g\n',traffic)]; % Write the port data in a Verilog readable format
fprintf(fid4,txt4);   % write the Port text to the file

if traffic==0  % No packet on this input port so Leave function
   %disp('-----No Packet--------');
   X=[0];
   return
end


%------------- Begin PAC-------------------
d=traffic; %Step 504

Qd(d)=Qd(d)+1;

if (Qd(d)>p*sigma)  %Step 508 drop packet?
   Qd(d)=Qd(d)-1;  %Step 510
   %disp('-----Packet dropped @ 508--------');
```

```
   X=[0];
   return          %Return to step 502
end

if (Qd(d)==1)  %Step 512
   jd=SW_j; LC_j(d)=SW_j; % Step 514
   kd=SW_k; LC_k(d)=SW_k;
else
   jd=mod(LC_j(d),sigma)+1; %Step 516
   pkt=1;
   if (jd==1)  %Step 518
      kd=mod(LC_k(d),p)+1;  %Step 522
      LC_k(d)=kd;
      LC_j(d)=1;
   else
      kd=LC_k(d); %Step 520
      LC_j(d)=jd;
   end
end

%-------Step 524&525 3)Control Operations p664--------
ES_j= sum(ST(:,jd)==0); % Determine ES(j)
r=0;
for (out=1:NxN) % Determine(r)
   r=r+(Qd(out)<sigma);
end

if(Qd(d)>sigma)
   if(ES_j>r)
      ES_j=ES_j-1; %-1
   else
      %disp('-----Packet dropped @ 525--------');
      X=[0];
      return
   end

end
%-------End of step 524 & 525--------

%---------Step 526  Assign id-----------

% Recall id = 1 (set above)  coming into this routine
SearchCounter=0;
while ( SearchCounter < m )        %Dr.Kumars' Experimental id assignment Method(a)
   if ( (ST(id,jd) ==0) & (temp(id) ==0) ) % temp arrays indicates whether i th memory module has
                                  % already packet to forward
      ST(id,jd)=kd;            % write Kd to available i th memory module corresponding
                           % Jd location
      temp(id)=1;              % update temp i th array for future examination
      %id=(di+1) MOD m;         % increment i value with MOD fashion
      %X=[id jd kd d];
      X=encode([id jd kd d]); % Put i,j,k,d values into packet (each var gets 8 bits)
      return;                  %
    end
     id=mod(id,m)+1;            %Available location has not found, increment i value
     SearchCounter=SearchCounter+1;           % increment counter for while loop
```

```
 end                    % end of first while loop

% id should be 1 if it makes it here
while(ST(id,jd)>0)  %Dr.Kumar's Experimental id assignment Method(b) (catches rare case of 2 equal
id val's)
   id=mod(id,m)+1; % On the first loop id will be 2
   if (id==1)    % Make sure we don't have a full OSV (looped back)
      disp('-OSV is now full, This case not handled yet--');
      X=[0];
      return
   end
end
ST(id,jd)=kd;
%-------end Step 526----------


X=encode([id jd kd d]);  % Put i,j,k,d values into packet (each var gets 8 bits)
return
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University


function oini=RO(reset,action) % needs a reset signal because of counter (switch_cycle)
% RO                   %   Hardware may just use universal switch_cycle counter
% Memory module "read operation" algorithm
%   Reads data in the mem_array to the Output Interconnection Network Input (oini)

global m sigma NxN p
global mem_array OSA
global SW_j SW_k    % so the write_file function may use it to display the window
persistent switch_cycle


if(~reset) % active low reset
   switch_cycle=-2;    % Creates an independent SW counter with an offset of 3
   oini=zeros(m,4);    % Output is 0's (not used by anything yet)
```

```
   SW_j=1; SW_k=1;     % Step 402
   return
end

% else reset==1

oini=zeros(m,1);    % Stuff output with 0's

% else switch_cycle >= 1


if (action==1)    % Perform read operation
   if (switch_cycle<1)
      return
   end
   for module=1:m
      dat=mem_array(module,SW_j);     % extract data from the slot
      kd=floor(mod(dat,2^24)/2^16);   % mask the bits for kd data item
      if OSA(module,SW_j)==SW_k   % Step 902 & 904
         oini(module)=mem_array(module,SW_j);    % Output the packet         % Step 906
         mem_array(module,SW_j)=0;   % Clear the read memory location for (viewability)
         OSA(module,SW_j)=0;         % Clear the OSA to make room for new data   % Step 906
      else
         oini(module)=0; % Output a 0 if there is no packet present
      end
   end

else    % Other wise update variables according to SW scheme
   if (switch_cycle>0)
      SW_j=mod(switch_cycle, sigma)+1; % Step 410

      if(SW_j==1)                % Step 412
         SW_k=mod(SW_k, p)+1;     % Step 414
      end
   end
   switch_cycle=switch_cycle+1; %Update counter (Step 404)

   return
end
```

---

```
%(c) 2004 B.R. Phelps, North Carolina State University


function mem_array=WO(mem_port_c1, mem_port_c2)
% WO
% Memory module "write operation" algorithm
%   Stores data in the memory module array
% The Algorithm Requires 2 writes per switch cycle: mem_port_c1 & mem_port_c2

global m sigma NxN p
global mem_array OSA  % These are declared global b/c they are read by many functions


% First cycle of packets
for module=1:m
```

```
   %--------input module 30_k1 (extracts j & k)-----------
   [id,jd,kd,dd]=decode(mem_port_c1(module)); % Step 850

   %--------input module end-----------

   if kd % If there is a packet
      % Stuffs id,jd,kd,dd into 1 space
      mem_array(module,jd)=mem_port_c1(module); % Put IIN(cycle 1) values into memory (each var
gets 8 bits)
      OSA(module,jd)=kd;     % Step 854   Write to OSA
   end
end

% Second cycle of packets(usually not any unless there are identical id's assigned)
for module=1:m

   %--------input module 30_k1 (extracts j & k)-----------
   [id,jd,kd,dd]=decode(mem_port_c2(module)); % Step 850
   %--------input module end-----------

   if kd % If there is a packet
      % Stuffs id,jd,kd,dd into 1 space
      mem_array(module,jd)=mem_port_c2(module); % Put IIN(cycle 2) values into memory (each var
gets 8 bits)
      OSA(module,jd)=kd;     % Step 854   Write to OSA
   end
end


return
```

---



```
function write_file(action,packet,packet_out);
% Writes to a file 'test.dat' in a viewable "cycle by cycle" method for easy viewability
% Assumes the display for the editor is set to a monospaced font (Matlab default)
% Manipulate the file based on the following parameters:
persistent fid fid2 fid3 cycle txt txt2 txt3
global NxN m p sigma mem_array SW_j SW_k fid4 txt4
% Constants to classify action
DEBUG=0;
open_file=1;
write_next=2;
close_file=3;
```

```
if(action==open_file)
   cycle=0;                    % Initialize the cycle counter

          if(DEBUG)
          fid=fopen('test_memory.dat','w');     % Open the file for data
                    txt=sprintf('Packets written is the Output of PAC delayed for easy
viewing\n\nMemory cycle by cycle in the format: i/j/k/d\n');
          end
   fid2=fopen('test_output.dat','w');     % Open the file for data
          %fid3=fopen('verilog_mem.dat','w');     % Open the file for verilog data

   %txt2=sprintf('Output ports cycle by cycle in the format: "port_number:  i/j/k/d"\n');
          %fprintf(fid2,txt2);   % write the Output Port text to the file
          %txt3=''; % Verilog txt strings
end

if(action==write_next)

   cycle=cycle+1;

          if (DEBUG)% commented out with an if to avoid a large txt array for long runs
          txt=[txt sprintf('\nCycle: %g\nPackets written:\n',cycle)];
          % Display the packets currenly being read
          for k1=1:NxN
           [id,jd,kd,dd]=decode(packet(k1));
                              txt=[txt sprintf('%g/%g/%g/%g\n',id,jd,kd,dd)];


          %  if (cycle>2)
                    %txt3=[txt3 sprintf('%g\n%s\n%g\n%g\n',id,dec2hex(jd),kd,dd)]; % Write the input
port
          %end
          end
          % Loop through the memory & display each location
          for module=0:m
           for slot=0:sigma

          if (module==0)      % on the first file write cycle
                    if (slot>0)       % Print the slot number
                     if((slot==SW_j)&(cycle>=4))
                    txt=[txt sprintf('[ SW_k=%g ]\t',SW_k)]; % This slot is being read, Display the SW_k
                     else
                    txt=[txt sprintf('\t%g\t\t',slot)];     % Print just the slot #
                     end
                    else
                     txt=[txt sprintf('\t')];      % otherwise insert tabs
                    end
          else % module number is > 0)
                    if (slot >0)
                     dat=mem_array(module,slot);     % extract data from the slot

                     [id,jd,kd,dd]=decode(dat);      % decode the data from the slot if its there


                     if id>0
                    txt=[txt sprintf('%g/%g/%g/%g \t',id,jd,kd,dd)]; %put into text array
```

```
                    else
                   txt=[txt sprintf(' / / / \t\t')];   % put 'empty slots' into text array
                    end
                   else
                    txt=[txt  sprintf('%g\t',module)];      % Output the module # for labeling
                   end

          end
           end
           txt=[txt sprintf('\n')];        % EOL character insetion
          end
          end

          % Commmented out to save space
   %txt2=[sprintf('\nCycle: %g\nPackets on port Output:\n',cycle)];
          %fprintf(fid2,txt2);   % write the Output Port text to the file
   for k1=1:NxN
      [id,jd,kd,dd]=decode(packet_out(k1));
      txt2=[sprintf('%g/%g/%g/%g\n',id,jd,kd,dd)];    % Display the port,id,jd,kd,dd
           fprintf(fid2,txt2);   % write the Output Port text to the file

          end

          fprintf(fid2,'\n');   % write the Output Port text to the file
end


if(action==close_file)


          %fprintf(fid3,txt3);   % write the Port text to the file
          if(DEBUG)
                   fprintf(fid,txt);   % write the Memory text to the file Comment out for large runs!
          fclose(fid);        % close the file
          end
          fclose(fid2);        % close the file
  % fclose(fid3);        % close the file
          fclose(fid4);        % close the file
          %clear txt3;
    clear txt4;          % deallocate the txt memory
   clear txt; clear txt2;          % deallocate the txt memory
end
```

---

## Verilog For The 1024-bit Priority Encoder

```
module priority_enc (clock, reset, data, PE);


/*-----------Inputs-------------------------------*/

input       clock, reset;  /* clock */
input [1023:0] data;  /* input initial count */

/*-----------Outputs------------------------------*/
```

```
output[9:0]      PE;  /* zero flag */

/*----------------Nets and Registers----------------*/
/*---(See input and output for unexplained variables)---*/

reg [1023:0] value;       /* current count value */
reg [9:0]   PE;
reg [9:0]
      result1,result2,L00,L01,L02,L03,L04,L05,L06,L07,L10,L11,L12,L13,L20,
L21;
integer k;

//
always@(data)
      value=data;
//

always @ (value[511:0])
      begin
      result1=10'b1111111111;
      for (k=511; k>=0; k=k-1)
            if(!value[k])
                  result1=k;
      end

always @ (value[1023:512])
      begin
      result2=10'b1111111111;
      for (k=1023; k>=512; k=k-1)
            if(!value[k])
                  result2=k;
      end

// Count Flip-flops with input multiplexor
always@(posedge clock)
  begin  // begin-end not actually need here as there is only one
statement
    if (!reset)
            begin
                  //value<=0;
                  PE <= 0;
      end
       else
            begin
                  //value <= data;
            PE <= (result1==10'd1023) ? result2 : result1;
            end
  end



endmodule
```

## Synthesis Script for 1024-bit Priority Encoder

```
Read -f Verilog PE1024.v

target_library = {"ncsulib25_worst.db"}
link_library = {"ncsulib25_worst.db"}


Create_clock -period 12000 -waveform {0 6000} clock
set_clock_skew  -uncertainty 300 clock

set_input_delay 1100 -clock clock all_inputs() - clock

set_output_delay 950 -clock clock all_outputs()


set_driving_cell -cell "dp_2" -pin "q" all_inputs() – clock

port_load = 0.5 + 3 *  load_of (ncsulib25_worst/dp_2/ip)
set_load port_load all_outputs()


set_max_area 0

replace_synthetic -ungroup

check_design
check_timing

compile -map_effort medium -verify -verify_effort medium


report_timing
target_library = {"ncsulib25_best.db"}
link_library = {"ncsulib25_worst.db"}
translate


set_fix_hold clock
compile -only_design_rule -incremental


report_timing -delay min

write_timing -output pe_min.sdf -format sdf


target_library = {"ncsulib25_worst.db"}
link_library = {"ncsulib25_worst.db"}
translate

report_timing

write -f verilog -o pe_final.v

write -hierarchy -format verilog -output pe_heirarchy.v


write_timing -output pe_max.sdf -format sdf
```

```
report_cell
```

## Final Synthesis Results for 1024 bit Priority Encoder

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : priority_enc
Version: 2001.08
Date   : Sat Mar 27 15:51:27 2004
****************************************

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: data[1000] (input port clocked by clock)
  Endpoint: PE_reg[3] (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clock (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.00       0.00
  input external delay                  1100.00    1100.00 r
  data[1000] (in)                         84.29    1184.29 r
  U10869/op (and2_2)                     186.50    1370.80 r
  U14016/op (nand3_2)                    213.05    1583.85 f
  U15677/op (inv_2)                      103.63    1687.48 r
  U15184/op (nand2_2)                    117.66    1805.14 f
  U11602/op (nor3_2)                     218.11    2023.25 r
  U11603/op (nor2_2)                      99.39    2122.64 f
  U11604/op (nor3_2)                     224.70    2347.34 r
  U11605/op (nor3_2)                     116.59    2463.93 f
  U11606/op (nor2_2)                     159.01    2622.93 r
  U11607/op (nor2_2)                      90.12    2713.06 f
  U11608/op (nor3_2)                     226.57    2939.62 r
  U11609/op (nor2_2)                      76.59    3016.21 f
  U11610/op (nor3_2)                     220.67    3236.88 r
  U11611/op (nor2_2)                     100.95    3337.82 f
  U11612/op (nor3_2)                     232.14    3569.97 r
  U11613/op (nor2_2)                     102.64    3672.61 f
  U11614/op (nor3_2)                     233.89    3906.49 r
  U11615/op (nor2_2)                     102.72    4009.21 f
  U11616/op (nor3_2)                     228.08    4237.30 r
  U11617/op (nor3_2)                     110.23    4347.53 f
  U11618/op (nor3_2)                     233.05    4580.58 r
  U11619/op (nor2_2)                     104.17    4684.75 f
  U11620/op (nor2_2)                     137.85    4822.60 r
  U11398/op (or2_1)                      142.89    4965.50 r
  U11397/op (or3_1)                      181.28    5146.78 r
  U11621/op (nor2_2)                      86.14    5232.91 f
```

```
U11622/op (nor2_2)                       140.01    5372.93 r
U11623/op (nor2_2)                        90.67    5463.60 f
U11624/op (nor3_2)                       221.53    5685.13 r
U11625/op (nor2_2)                       105.58    5790.71 f
U11626/op (nor2_2)                       152.84    5943.56 r
U11627/op (nor3_2)                       101.37    6044.93 f
U11628/op (nor3_2)                       232.32    6277.25 r
U11629/op (nor2_2)                        75.69    6352.94 f
U11630/op (nor3_2)                       213.32    6566.26 r
U11631/op (nor3_2)                       111.59    6677.85 f
U11632/op (nor3_2)                       238.58    6916.42 r
U11396/op (or3_2)                        134.07    7050.50 r
U11395/op (or3_1)                        176.45    7226.95 r
U11633/op (nor2_2)                        67.70    7294.66 f
U11634/op (nor3_2)                       216.98    7511.64 r
U11635/op (nor2_2)                       100.93    7612.57 f
U11636/op (nor3_2)                       238.08    7850.64 r
U11637/op (nor3_2)                        91.00    7941.65 f
U11638/op (nor2_2)                       145.22    8086.86 r
U11639/op (nor3_2)                        97.73    8184.60 f
U11640/op (nor3_2)                       216.52    8401.12 r
U11641/op (nor3_2)                       109.16    8510.28 f
U11642/op (nor3_2)                       232.12    8742.40 r
U11643/op (nor2_2)                       105.66    8848.06 f
U11644/op (nor3_2)                       242.82    9090.88 r
U11064/op (nor2_2)                        75.75    9166.63 f
U11063/op (nor3_2)                       224.69    9391.32 r
U11645/op (nor2_2)                        82.62    9473.94 f
U11646/op (nor2_2)                       148.42    9622.36 r
U11647/op (nor2_2)                        89.99    9712.35 f
U11648/op (nor3_2)                       226.42    9938.77 r
U11649/op (nor2_2)                        79.17   10017.94 f
U11650/op (nor2_2)                       138.21   10156.15 r
U11651/op (nor3_2)                        97.03   10253.18 f
U11652/op (nor3_2)                       223.62   10476.80 r
U11653/op (nor2_2)                       101.17   10577.97 f
U11654/op (nor3_2)                       232.37   10810.35 r
U11655/op (nor2_2)                       102.58   10912.93 f
U11656/op (nor3_2)                       239.60   11152.54 r
U11658/op (nor2_2)                        77.53   11230.07 f
U11404/op (or2_2)                        192.86   11422.93 f
PE_reg[3]/ip (dp_2)                        0.00   11422.93 f
data arrival time                                 11422.93

clock clock (rise edge)                12000.00   12000.00
clock network delay (ideal)                0.00   12000.00
clock uncertainty                       -300.00   11700.00
PE_reg[3]/ck (dp_2)                        0.00   11700.00 r
library setup time                      -237.23   11462.77
data required time                                11462.77
-----------------------------------------------------------
data required time                                11462.77
data arrival time                                 -11422.93
-----------------------------------------------------------
slack (MET)                                          39.83
```

## C++ Code

## Memory Simulation

```
//(c) 2004 B.R. Phelps, North Carolina State University
// stdafx.cpp : source file that includes just the standard includes
// Switch_Memory2.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
/*Reads Traffic0.txt.. trafficn.txt generated by matlab b_traffic
algorithm & my matlab Program*/

#include "sim.h"
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
int sim_switch(int args){
      const unsigned int MAX_TEXT = 10000;
      const unsigned int MAX_MEM =1000000;
      unsigned int num=0;
      unsigned int MEM_SIZE=64;
      unsigned int *mem = new unsigned  int[MAX_MEM];
      char text[MAX_TEXT];

      unsigned int NxN;
      unsigned int Eb;
      float Load;
      unsigned int packets;

      unsigned int indx;
      unsigned int dummy, round;
      unsigned int curr_traffic;
      unsigned int kx;
      unsigned int MAX_ROUNDS=0;
      char filename_r[20];


      printf("Simulation Running...\n");

      //FILE *fp_r = fopen("C:/Documents and Settings/administrator/My
Documents/Visual Studio Projects/Switch_Memory2/traffic/traffic.dat","r");
      //FILE *fp_w = fopen("C:/Documents and Settings/administrator/My
Documents/Visual Studio Projects/Switch_Memory2/traffic/results.txt","w");

      FILE *fp_w = fopen("results.txt","w");
      for(unsigned int k=0;k<MEM_SIZE;k++){           // Clear the memory
            mem[k]=0;
      }
      FILE *fp_r;
```

```
for(round=0; round<=MAX_ROUNDS; round++){       // Runs once unless the
simulator is told otherwise
      sprintf(filename_r,"traffic%d.txt",round);      //Generate the
filename for reading
      printf("%s\n",filename_r);
      fp_r = fopen(filename_r,"r");                   // open the
generated filename

      if(round==0){
            fgets(text,MAX_TEXT, fp_r);
            sscanf(text, "%d", &MAX_ROUNDS);    // MAX_ROUNDS becomes >0
if need be
      }

      fgets(text,MAX_TEXT, fp_r);   // Get the Parameters
      sscanf(text, "%d %g %d %d ", &NxN, &Load, &Eb, &packets);   // Print
out the packet parameters

      while(fgets(text,MAX_TEXT, fp_r)){  // Reads white space
                        //fgets(text,20, fp_r);
                        //sscanf(text, "%d", &curr_traffic);
                        for(unsigned int kk=0;kk < NxN; kk++){    // Port
loop
                              fgets(text,20, fp_r);         // Gets 1st
cycle of packets
                              sscanf(text, "%d", &curr_traffic);

                              //fprintf(fp_w,"Port in:
%d\n",curr_traffic);
                              if(curr_traffic>0){
                                    dummy = 1; kx=0;  // reset flag &
index

                                    while((mem[kx]!=0)&&(kx<MEM_SIZE)){ //
search for the empty sslot
                                          kx++;
                                    }
                                    if (mem[kx]==0){  // Fill the empty
slot if it exists
                                          indx=kx; dummy=0;
                                    }


                                    if(dummy==0){// fill empty slot
                                          mem[indx] = curr_traffic;
                                    }
                                    else{ // create more slots & fill next
empty slot

                                          mem[MEM_SIZE] = curr_traffic; //
put packet in new empty slot
                                          MEM_SIZE = MEM_SIZE+64;
      // increase the memory to accomodate new slot
                                          printf("%d\n",MEM_SIZE);
      // display the new size
                                    }
```

```
                        }
                    }

                    for(unsigned int kk=1; kk<=NxN; kk++){    //% read
out packets that match the current port number

                        dummy = 0; kx=0;  // reset flag & index
                        while((mem[kx]!=kk)&&(kx<MEM_SIZE)){
                            kx++;
                        }
                        if (mem[kx]==kk){       // mark the slot
that is the current port number
                            indx=kx; dummy=1;
                        }

                        if((dummy > 0)){//  % Read out a packet if
it is the current port number
                            mem[indx] = 0;
                        }

                    }

      }
      // Print the simulation statistics and parameters
      fprintf(fp_w,"NxN: %d\tLoad: %g\tBurstiness: %d\tPacket Amount:
%d\n", NxN, Load, Eb, packets);
      fprintf(fp_w,"Memory Required: %d\n", MEM_SIZE);
      fprintf(fp_w,"---------Next Round-----------\n");
      fclose(fp_r);     //close the file for this round so we can open the
next
}

      fclose(fp_w);
      return 1;
}
```

---

```
//(c) 2004 B.R. Phelps, North Carolina State University

//#ifndef SIM_H
//#define SIM_H
#include "stdafx.h"


int sim_switch(int args);

//#endif
```

---

```
//(c) 2004 B.R. Phelps, North Carolina State University

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//
```

---

```
//(c) 2004 B.R. Phelps, North Carolina State University

#include "stdafx.h"

#using <mscorlib.dll>

using namespace System::Reflection;
using namespace System::Runtime::CompilerServices;

//
// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
//
[assembly:AssemblyTitleAttribute("")];
[assembly:AssemblyDescriptionAttribute("")];
[assembly:AssemblyConfigurationAttribute("")];
[assembly:AssemblyCompanyAttribute("")];
[assembly:AssemblyProductAttribute("")];
[assembly:AssemblyCopyrightAttribute("")];
[assembly:AssemblyTrademarkAttribute("")];
[assembly:AssemblyCultureAttribute("")];

//
// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the value or you can default the Revision and Build
Numbers
// by using the '*' as shown below:

[assembly:AssemblyVersionAttribute("1.0.*")];

//
// In order to sign your assembly you must specify a key to use. Refer to
the
// Microsoft .NET Framework documentation for more information on assembly
signing.
//
// Use the attributes below to control which key is used for signing.
//
// Notes:
//   (*) If no key is specified, the assembly is not signed.
//   (*) KeyName refers to a key that has been installed in the Crypto
Service
//       Provider (CSP) on your machine. KeyFile refers to a file which
contains
//       a key.
//   (*) If the KeyFile and the KeyName values are both specified, the
//       following processing occurs:
```

```
//       (1) If the KeyName can be found in the CSP, that key is used.
//       (2) If the KeyName does not exist and the KeyFile does exist, the
key
//           in the KeyFile is installed into the CSP and used.
//   (*) In order to create a KeyFile, you can use the sn.exe (Strong
Name) utility.
//        When specifying the KeyFile, the location of the KeyFile should
be
//        relative to the project directory.
//   (*) Delay Signing is an advanced option - see the Microsoft .NET
Framework
//       documentation for more information on this.
//
[assembly:AssemblyDelaySignAttribute(false)];
[assembly:AssemblyKeyFileAttribute("")];
[assembly:AssemblyKeyNameAttribute("")];
```