<u>CHAPTER 5</u>

SIMULATION AND IMPLEMENTATION OF CROSSBAR SWITCHES WITH SCHEDULING ALGORITHMS

5
SIMULATION AND IMPLEMENTATION OF CROSSBAR SWITCHES WITH SCHEDULING ALGORITHMS

5.1 INTRODUCTION

Matrix-like space division switch fabric includes controller hardware that handles port contention by ensuring that only one input port accesses each output port at any time. There is a centralized scheduler in an N×N switch that considers requests from all the input queues and determines the best realizable input to output mapping for the crossbar during each time slot by using a scheduling algorithm. This scheduling algorithm has to be fast, efficient (good throughput), easy to implement in hardware, fair in serving all the inputs, low latency, and QOS support. We start with a brief introduction of scheduling algorithms like PIM, RRM, iSLIP, RPA and DPA and detail their features. Then we simulate 4x4, 8x8, 16x16 and 32x32 input buffer crossbar switches with above scheduling algorithms. At last we implement, 4x4 and 8x8 ATM crossbar switch, in VHDL using ALTERA's MAXPLUS II /QUARTUS software and compare above switches with above scheduling algorithms based on simplicity of implementation, area requirement on the same platform.

5.2 STANDARD SCHEDULING ALGORITHMS FOR INPUT QUEUE SWITCH

In general, scheduling algorithms can be divided into two areas: maximum matching and maximal weighted matching. An algorithm that finds the maximum number of matches between inputs and outputs and provides the highest possible throughput in each slot for an input-queued switch is known as maximum matching algorithm. Maximum match takes too long time to compute and converge and starve some connections. Maximal matching algorithm, iteratively adds connections to fill in the missing connections left by the previous iteration, because connection made in the previous iteration may not be removed. It achieves a close approximation to maximum for many traffic patterns.[ref17compa]

A maximum matching algorithm finds a match with the maximum size or weight, called maxsize and maxweight matching respectively. (maxweight is maxsize if the weight on each of the edges is unity.)

5.2.1 Maximum Size Matching: Scheduling algorithm by McKeown, Anantharam, and Warland [67] attempts to maximize the number of connections made in each cell time, and hence maximize the throughput by connecting maximum number of edges. If the traffic is independent and identically distributed (i.i.d.) arrivals and uniformly distributed among all the VOQ's, the algorithm will result in a higher throughput. This algorithm is stable and achieves 100% throughput for independent uniform traffic but could lead to starvation and hence queue overflow or instability, if the arrival processes are not uniform [67]. It does not consider the backlog of cells in the VOQ's or the cells that have been waiting in line to be served so cause a reduction in throughput for non-uniform traffic. This algorithm is too complex to implement in hardware and takes too long to complete [68]. The best known maximum size matching algorithm converges in O ($n^{5/2}$) time [65].

5.2.2 Maximum Weight Matching: This algorithm assigns a weight to each input queue. The matching algorithm finds an input-output match that has the highest sum of weights. This algorithm is stable for both uniform and non-uniform traffic [67]. The weight assigned to each queue is usually equal to the occupancy of the queue and therefore the longest queue has the highest weight. Hence this algorithm is also called Longest Queue First (LQF).It needs multi-bit comparators to compare the weights of the queues and hence complexity is high i.e., O (N^3logN).

5.2.3 Oldest Cell First (OCF): This algorithm uses the waiting times of cells as requesting weights and selects a match such that the sum of all queue waiting times is maximized. It has high complexity i.e., O (N^3logN) and difficult to implement in hardware [69].

5.2.4 Longest Port First (LPF): Algorithm by McKeown is a variation of the LQF scheme [68]. In LQF algorithm, each queue has a weight equal to the length of the queue. In LPF, however, the weight (also called port occupancy) of each queue is the sum of aggregate input and output queue occupancies. This algorithm finds the match that is both maximum size and maximum weight. The complexity of the LPF scheme is $O(N^{2.5})$, and can be implemented in hardware.

5.2.5 Parallel Iterative Matching (PIM): PIM algorithm was developed by DEC system research centre. It is based on randomness (to avoid starvation) and iteration [65]. This algorithm converges on a conflict-free match in multiple iterations. All inputs and outputs that have not been match in previous iterations are eligible for matching in the next iteration. There are three steps in each iteration and operate in parallel on each input and output as follows:

a. Request: All unmatched inputs send requests to every output for which they have queued cell;

b. Grant: If an unmatched output receives any request, it grants at random one of its requesting input;

c. Accept: If an input receives a grant, it accepts it but if it receives multiple grants then it accepts one by selecting an output randomly among those that granted its request.

These three steps are repeated for the inputs that are not paired with any outputs, until they converge to a maximal match. A maximal match is one in which each node is either matched or has no edge to an unmatched node. PIM can be viewed as an algorithm that finds a matching in a N^2 X N bipartite graph (because there are N queues at each input, one for each output) with independent arbiters at each input and output port making decisions randomly in step 2 and 3.

Advantages: 1. In each iteration of random matching, a minimum average of 3/4 of the remaining possible connections are matched or eliminated. Therefore this algorithm converges to a maximal match in an average of O (log N) iterations.

2. It ensures that all requests are eventually granted.

Disadvantage: 1. It has large queuing latency in the presence of heavy traffic load. 2. It is expensive and difficult to implement in hardware. 3. It can lead to unfairness between connections and the multiple iterations are time consuming. 4. Inability to provide prioritized QoS.

5.2.6 Round Robin Matching (RRM): RRM is perhaps the simplest and most obvious form of iterative round-robin scheduling algorithms, comprising a 2-D array of round-robin arbiters; cells are scheduled by round-robin arbiters at each output, and at each input. The three steps of RRM arbitration as shown in figure 5.1 are:

Step 1: Request. Each input sends a request to every output for which it has a queued cell.

Step 2: Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input.

Step 3: Accept. If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output.

RRM algorithm removes the unfairness and complexity inherent in the PIM algorithm. The algorithm performs well on a single iteration and converges to a maximal match in an average of *O (log N)* iterations.

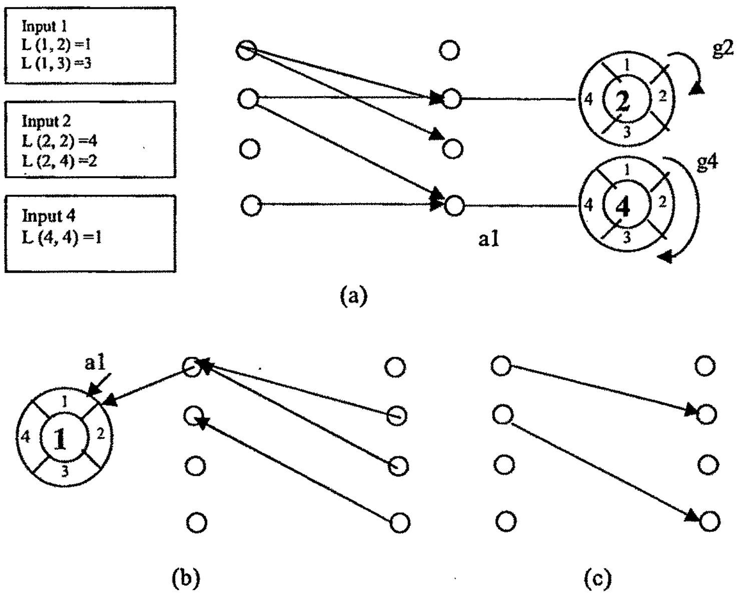


Figure 5.1: RRM scheduling algorithm (a) Request, (b) Grant, and (c) Accept.

The reason for the poor performance of RRM lies in the rules for updating the pointers at the output arbiters. So, the RRM algorithm performs poorly under heavy traffic due to a synchronization phenomenon [65]. RRM does not perform well, but it helps us to understand how iSLIP performs.

5.2.7 iSLIP: iSLIP is an iterative algorithm, achieved by making a small variation to the RRM scheme [65]. The iSLIP algorithm improves upon RRM by reducing the synchronization of the output arbiters. Request step and accept step of iSLIP are same as RRM, the only difference in grant step is that iSLIP does not move grant pointers unless the grant is accepted.

Step 2: Grant. If an output receives any requests, it chooses the one that appears next in a fixed round-robin schedule, starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if, and only if, the grant is accepted in Step 3.

Those inputs and outputs not matched at the end of one iteration are eligible for matching in the next. This small change to the RRM algorithm makes iSLIP capable of handling heavy loads of traffic and eliminates starvation of any connections. The algorithm converges in an average of $O\ (log\ N)$ and a maximum of N iterations. iSLIP can fit in a single chip and is readily implemented in hardware[64].

This small change to the algorithm leads to the following properties of iSLIP with one iteration:

Property 1: Lowest priority is given to the most recently made connection. This is because when the arbiters move their pointers, the most recently granted (accepted) input (output) becomes the lowest priority at that output (input).

Property 2: The algorithm should not allow a nonempty VOQ to remain unserved indefinitely. This is because an input will continue to request an output until it is successful. The output will serve at most N-1 other inputs first, waiting at most N cell times to be accepted by each input. Therefore, requesting input is always served in less than N^2 cell times.

Property 3: Under heavy load, all queues with a common output have the same throughput. This is a consequence of Property 2: the output pointer moves to each requesting input in a fixed order, thus providing each with the same throughput [65].

The iSLIP algorithm uses rotating priority ("round-robin") arbitration to schedule each active input and output in turn. The main characteristic of iSLIP is its simplicity; it is readily implemented in hardware and can operate at high speed.

5.2.8 Wave Front Arbiter (WFA) (RPA-DPA)

WFA is a fair crossbar scheduler with a round robin priority rotation. The scheduling algorithm is based on a small combinational logic arbiter cell assigned to each input/output pair. When there is a request to send packets from a certain input port to a certain output port, the corresponding arbiter cell receives a request from the input. The arbiter then issues a grant for the requested output based on both the position of the priority round robin, and the grants issued to higher priority cells.

5.2.8.1 Rectilinear Propagation Arbiter (RPA)

In 4x4 two-dimensional ripple carry arbiter, bold cells are cells with request and shaded cells are cells that have received grant as shown in figure 5.2(a) and (b).

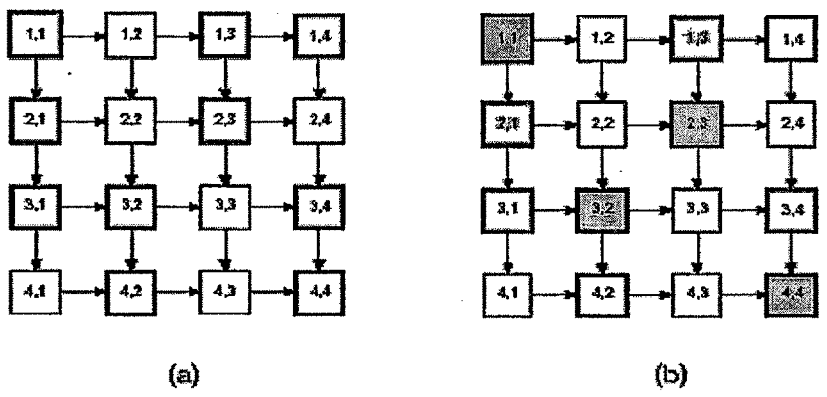
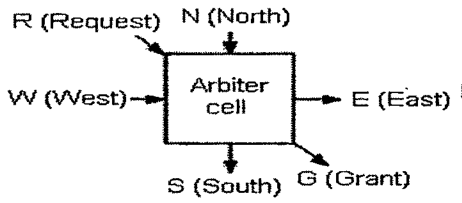


Figure 5.2: Two dimensional ripple-carry arbiter (request) (b) Two dimensional ripple-carry arbiter (grant)

Here, the rows correspond to the input ports and the columns correspond to the output ports of the switch. The arbiter is built from modular cells. A modular arbiter cell is shown in figure 5.3, with its internal combinational logic is shown in Figure 5.4. The label pairs *i, j* written on each cell specify that the cell is responsible for handling packets destined to go from input port *i* to output port *j*.

Input signal *R (Request)*, to every i, j arbiter cell is active when there is a packet destined for output port *j* at the head of the input port *i* buffer. This means that there is a packet at the head of queue *j* of input port module *i*.

Output signal *G (Grant)*, from every *i, j* arbiter cell, is active when the request from input port *i* to output port *j* has been granted by the scheduler and the arbiters on the top and left have not issued a grant.



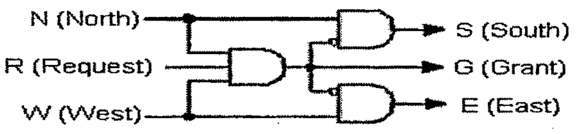


Figure 5.3: The RPA arbiter cell   **Figure 5.4 Internal combinational logic of RPA cell [51]**

-Since each input can be sending (and each output can be receiving) only one packet at a time, there should never be two or more granted requests in each row (and each column). For instance, having two requests granted in the same column at one time, causes that the output port corresponding to that column to receive two packets simultaneously. To ensure that this problem never occurs, signals *N (North)*, *S (South)*, *W (West)*, and *E (East)*, shown in Figure 5.3, are introduced. These signals in each cell have the duty of relaying to the next cell, or receiving from the former cell. In the ripple-carry architecture the *E* signal of every arbiter cell is connected to the *W* signal of the cell on its right. Similarly, the *S* signal of every arbiter cell is connected to the *N* signal of the cell on its bottom. (The *W* signal of cells in the first column and the *N* signal of cells in the first row are always set to logic one. The *S* signal of the cells in the last row and the *E* signal of the cells in the last column are floating). The logic circuit of figure 5.4 shows that whenever a *Grant* signal is high for a cell, signals *South* and *East* are forced to logic low, so that the cells on the right and bottom are never able to issue grants.

The arbitration process in the architecture of figure 5.2 is based on the following steps:

1) Start from the top left most cell (i.e. 1, 1);
2) Once any cell is reached, move to its right and bottom cells
3) For each arbiter cell, the *G (Grant)* signal is activated if and only if the *R (Request)* signal is active and there has not been any requests granted in the cells at the top and to the left;
4) If a request is granted, activate the *E (East)* and *S (South)* signals.

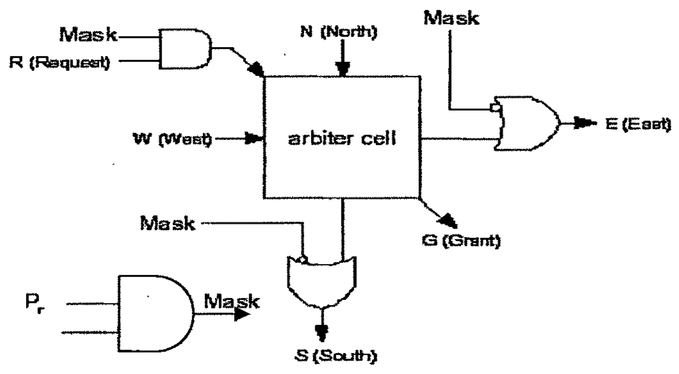


Figure 5.5 Arbitration cell for RPA

As shown in figures 5.5, arbitration cell is used as the basic building block. As shown in figure 5.4, each arbitration cell has three inputs (North, West, and Request) and three outputs (South, East, and Grant). Nij indicates that none of the rows above have been given any grant for output port j. Wij indicates that none of the columns before for input port i have been given any grant.

Gij = Rij ∩ Nij ∩ Wij

Sij = Nij ∩ NOT(Gij)

Eij = Wij ∩ NOT(Gij)

All the N inputs in row 1 and N inputs in column 1 are set to 1 [96].

The ripple-carry design gives the priority to the cells that are higher and to the left. It creates the issue of unfairness. Specifically, it gives the highest priority to cell (1, 1). Optimally one should be able to rotate the priority so that every cell has the chance of being the highest priority cell. One solution to this problem could be to make a cyclic architecture by connecting the South signals of the cells in the last row to the *North* signals of the cells in the first row. Similarly, the *East* signals of the last column have to be connected to the *West* signals of the first column as shown in figure 5.6.

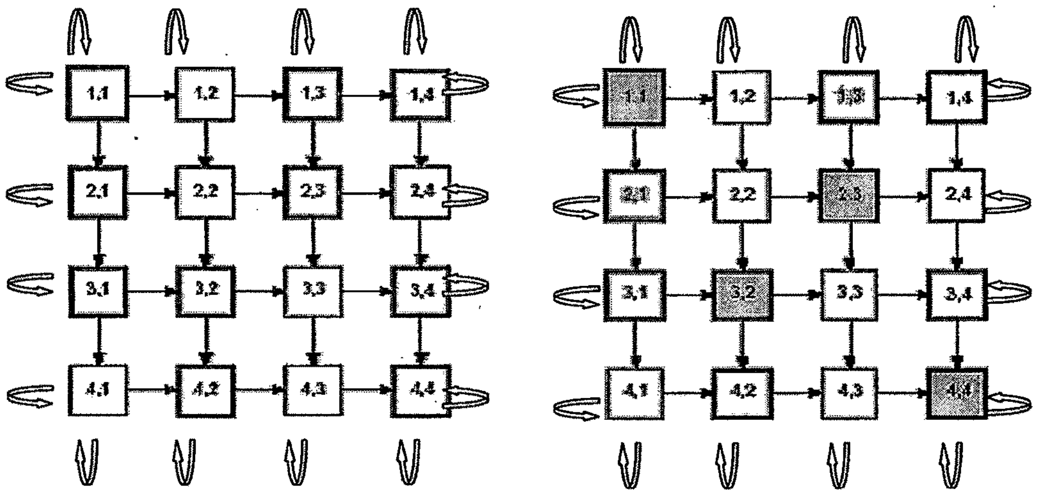


Figure 5.6 (a) A cyclic two-dimensional ripple carry arbiter architecture (b) Selected cells (in the shaded squares) with highest priority cell (1,1).

Such architecture would be fair because every cell can have the opportunity to be the highest priority cell. However, this architecture suffers from "combinational feedback loop" problem. Such architectures are difficult to design; they are not very well supported by logic synthesis tools and they have to be carefully simulated at the physical layout level. To overcome the cyclic feedback problem and to rotate the priorities, Hurt et al. have found a solution. At every time slot only n^2 cells (marked by the $n \times n$ bold window shown in figure 5.7and 5.8) are active. We call the bold window "the active window". In RPA, arbitration process begins at cell (1, 1) – top left of the architecture mask and ends at cell (4, 4) – bottom right of the mask as shown in figure 5.8. To provide equal opportunities to all the cells, we need to rotate priority round robin, first column wise and then row wise. We need two vectors Pr and Qr to rotate priority column and row wise. In figure 5.7, Pr = 1111000 and Qr = 1111000 to give highest priority to cell (1, 1). To give highest priority to cell (1, 2), set Pr = 1111000 and Qr = 0111100 as shown in figure 5.8. Highest priority cell switches from (1,1)→ (1,2)→ (1,3)→ (1,4)→ (2,1)----- at last (4,4) as each time slot progresses. Bold square indicates that the corresponding cell (i, j) has been requested ($R_{ij} = 1$) i.e. a request from input port i to transfer a cell to output port j. If request of the particular cell (i, j) is granted, it will block input port i and output port j for further requests. Granted cells are shown as dark squares.

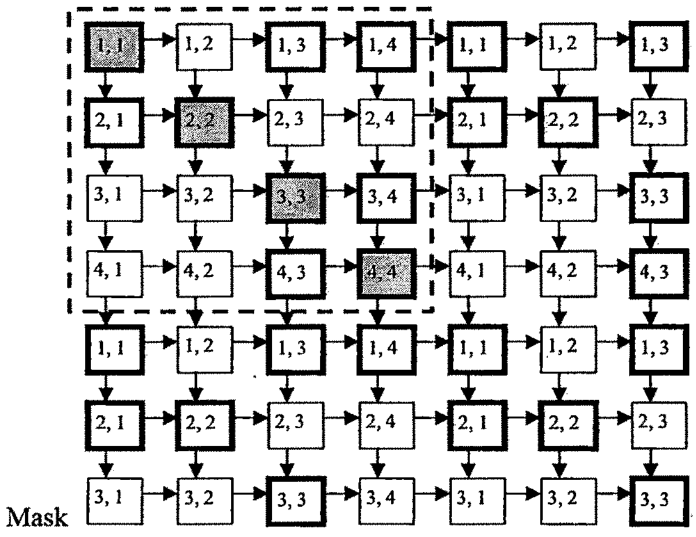


Figure 5.7 RPA architecture with highest priority cell (1, 1).

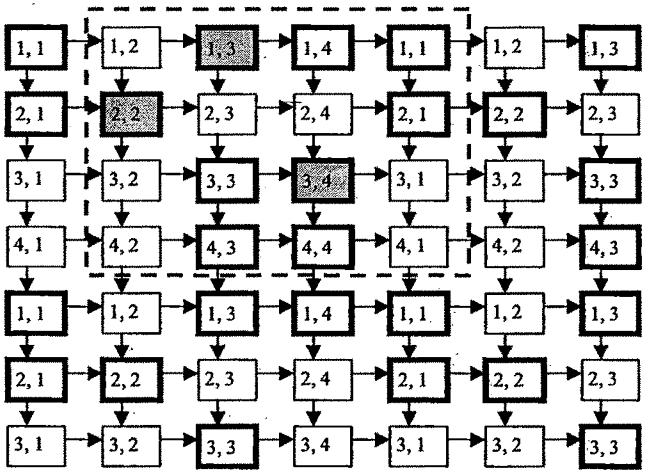


Figure 5.8 RPA architecture with highest priority cell (1, 2).

When the highest priority cell is (1, 1) as shown in figure 5.7, four grants were issued to cells (1, 1), (2, 2), (3, 3) and (4, 4). But, when the highest priority cell is (1, 2) as shown in figure 5.8, three grants were issued to cells (1, 3), (2, 2) and (3, 4). Assuming that each arbiter cell has a delay of *D*, for an n×n switch, the maximum arbitration delay through the whole switch is (2n-1) D, and then the time needed for realization of any permutation would be *(2n-1) D* for any *n x n* arbiter. J. Hurt, A. May, X. Zhu, and B. Lin have also introduced a modified version of the two-dimensional arbiter that has a shorter arbitration delay [51]. This new design called the diagonal propagation arbiter (DPA) is described in the next section.

5.2.8.2 Diagonal Propagation Arbiter (DPA) Architecture

As shown in figure 5.2, there are some cells in the two dimensional propagation arbiter that are independent of one another, in the sense that granting one of them does not prevent granting the others. The cells that are independent of one another are put in diagonal rows, as shown in Figure 5.9. Diagonal I consists of cells (1, 1), (4, 2), (3, 3) and (2, 4), which are independent of each other. Diagonal II consists of cells (2, 1), (1, 2), (4, 3) and (3, 4). Diagonal III consists of cells (3, 1), (2, 2), (1, 3) and (4, 4). Diagonal IV consists of cells (4, 1), (3, 2), (2, 3) and (1 4).

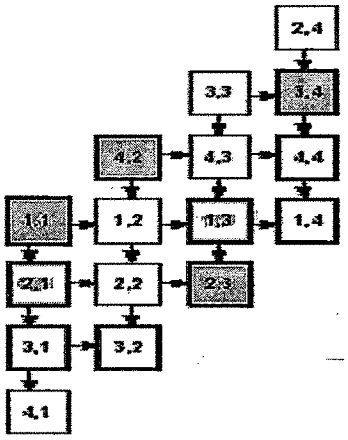


Figure 5.9 Diagonal Propagation Arbiter (DPA)

The arbitration process in the DPA architecture begins by considering the diagonal I. If there is a request for every cell in the first diagonal of Figure 5.9, they can all be granted. The cells with requests in the diagonal II will only receive grants if no cells on the top or on the left of them have yet received grants. In this design, the arbitration delay for an $n \times n$ switch is nD, D being the delay of a single arbiter cell. This is smaller than the delay in the RPA, which was $(2n-1) D$.

In this new architecture, shown in Figure 5.9, the first $(n-1)$ diagonals of an $n\times n$ DPA scheduler are repeated after the last row. The W signals of the first column and the N signals of the first diagonal are assigned to logic one.

The cells on the first diagonal inside the active window have the highest priority. The active window moves one step down in every time slot to rotate the priority. When the top most diagonal is diagonal n, the active window has traveled all the way through the DPA scheduler and, therefore, goes back to its starting position.

To implement priority rotations in this design, vector P is introduced. The $(2n-1)$ elements of vector P are named pr. They correspond to the $(2n- 1)$ diagonals of the scheduler in Figure 5.10. When the i^{th}

element of this vector is equal to 1, the i^{th} diagonal of the arbiter is active, (and resides in the active window). The algorithm for priority rotations is:

```
set P = "1111000".
if P = "0001111" then
        set P = "1111000"
else
      P= one bit right shift (P)
```

(This step is like moving the window one step down.)

As shown in figure 5.10, highest priority is given to the first diagonal. Figure 5.11 shows the arbiter cell of the rotating priority DPA. This arbiter is somewhat different from the basic arbiter cell introduced earlier. The difference is a signal called "*Mask*" (identical to the elements of vector *P*, *pr*) that indicates whether the arbiter cell is in the active zone. If the *Mask* input of a cell is logic 0, then there are no *Grants* given to that cell, and therefore, *E* and *S* signals shown in Figure 5.11 are forced to logic 1. The additional gates (one AND and two ORs) ensure that every request only takes effect if *Mask* is logic high. Figure 5.12 shows a similar example only with the highest priority given to the third diagonal.

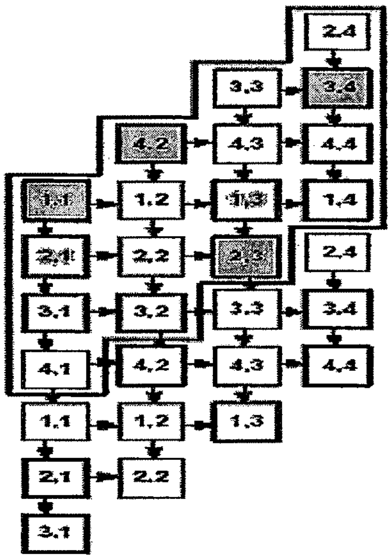


Figure 5.10: Diagonal Propagation Arbiter (DPA) with highest priority to the diagonal I.

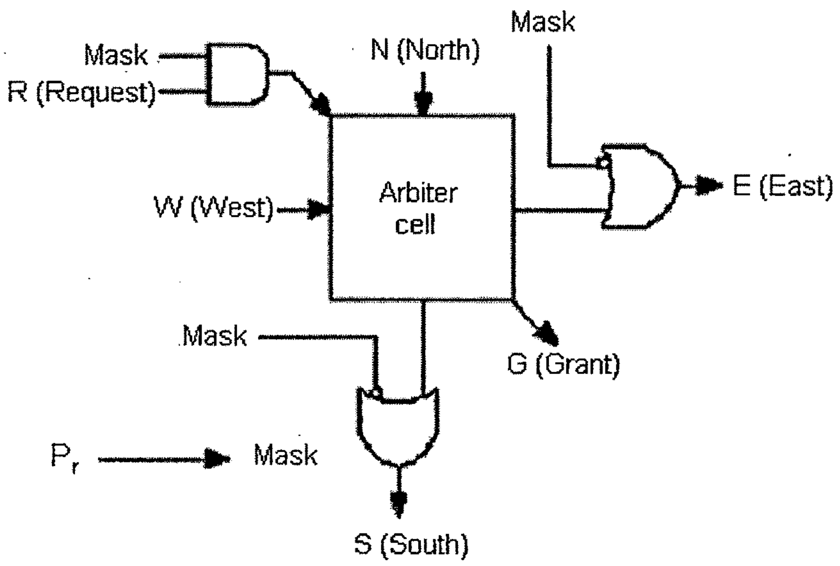


Figure 5.11: Modified arbitration cell for diagonal propagation arbiter (DPA) architecture [51].

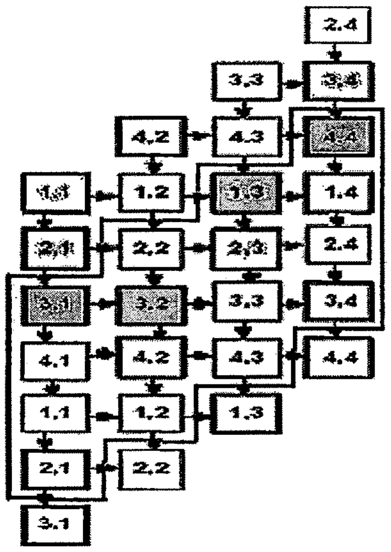


Figure 5.12: Diagonal Propagation Arbiter (DPA) with highest priority to the diagonal III.

5.3 SIMULATION AND COMPARISON OF DIFFERENT SCHEDULING ALGORITHMS

We have simulated PIM, RRM, iSLIP, RPA and DPA algorithms for 4x4, 8x8, 16x16, and 32x32 crossbar switches with four different traffic models (A,B,C and D) using MATLAB 7.0. Algorithms are simulated for 10000 time slots and results are taken by averaging the outcomes for 100 simulations for 4x4 and 8x8 switches. Algorithms are simulated for 1000 time slots and results are taken by averaging the outcomes for 10 simulations for 16x16 and 4 simulations for 32x32 switches and various parameters like throughput, average latency and delay variance have been measured for variation in offered load as well as variation in buffer size. For variation in offered load the buffer size is 2 in 4x4, 3 in 8x8 and 16x16 switches, while buffer size is 4 in 32x32 switches. For variation in buffer size the offered load is 90% for 4x4 switch and 80% for 8x8, 16x16 and 32x32 switches. All the scheduling algorithms run for one iteration only. Each data pattern from A to D generates its own data, which is applied to all the scheduling algorithms and the results are compared as shown below.

5.3.1 4x4 Switch Comparisons

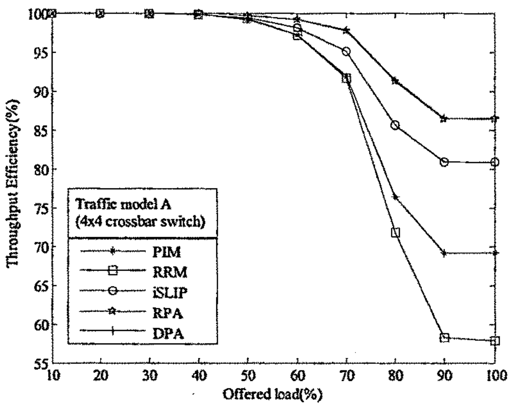


Figure 5.13 Throughput (%) v/s offered load

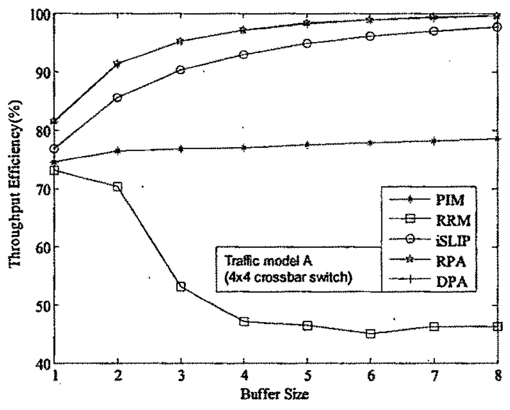


Figure 5.14 Throughput (%) v/s buffer size

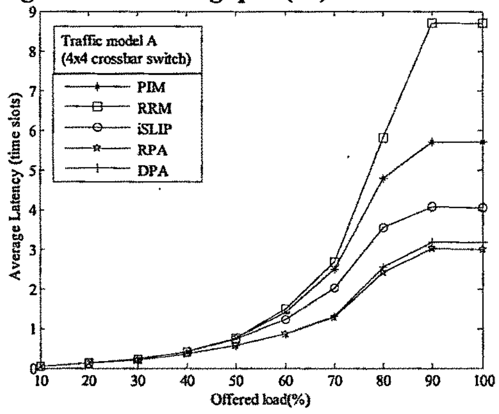


Figure 5.15 Average Latency v/s offered load

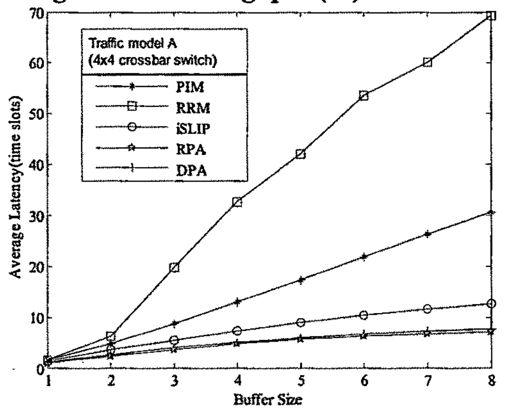


Figure 5.16 Average Latency v/s buffer size

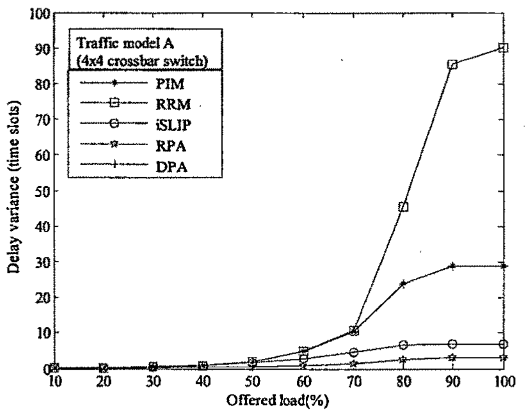


Figure 5.17 Delay variance v/s offered load

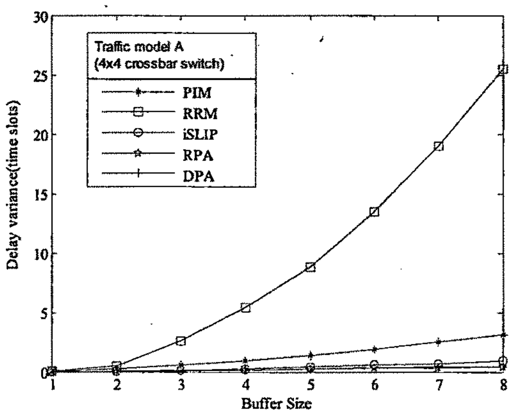


Figure 5.18 Delay variance v/s buffer size

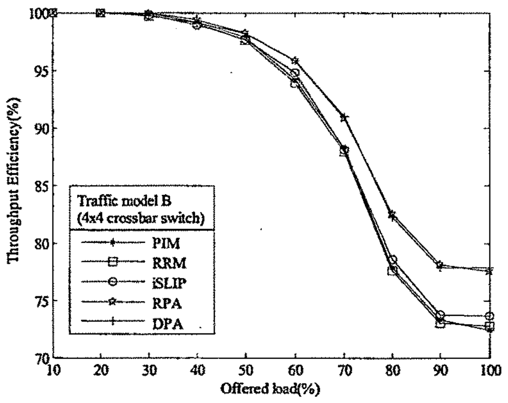


Figure 5.19 Throughput (%) v/s offered load

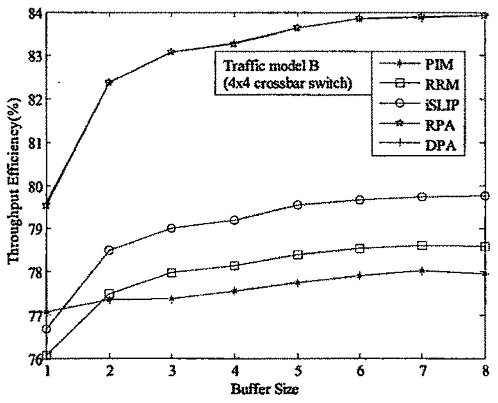


Figure 5.20 Throughput (%) v/s buffer size

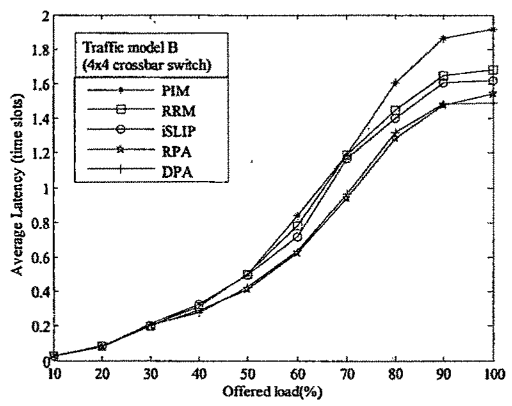


Figure 5.21 Average Latency v/s offered load

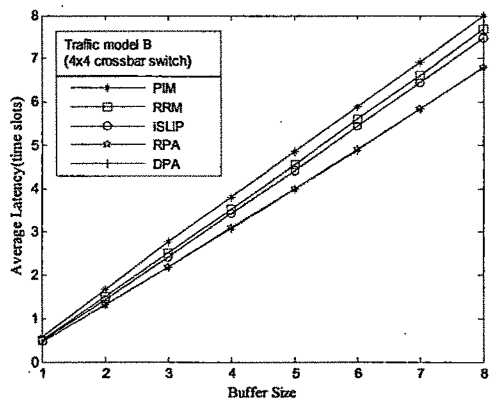
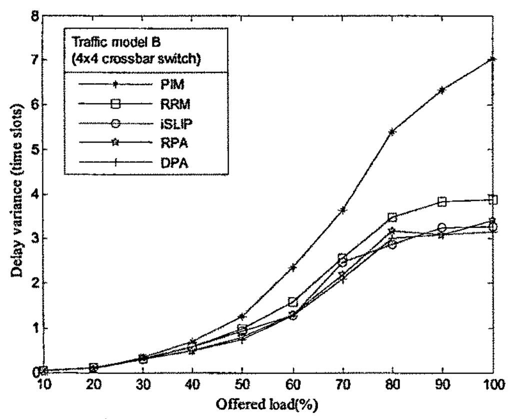


Figure 5.22 Average Latency v/s buffer size



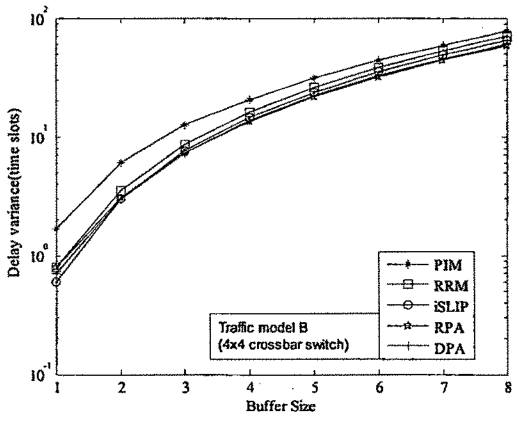


Figure 5.23 Delay variance v/s offered load **Figure 5.24 Delay variance v/s buffer size**

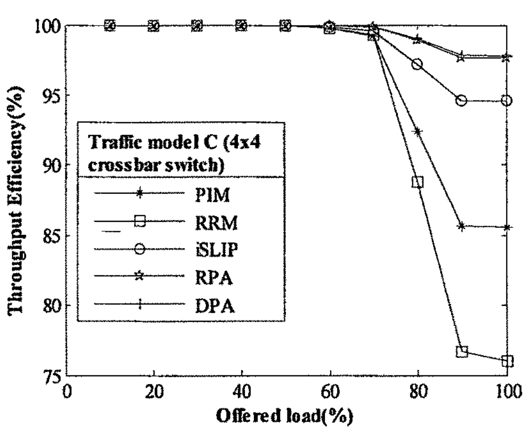


Figure 5.25 Throughput (%) v/s offered load

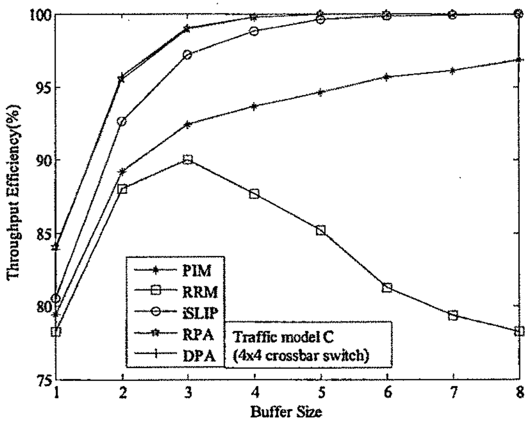


Figure 5.26 Throughput (%) v/s buffer size

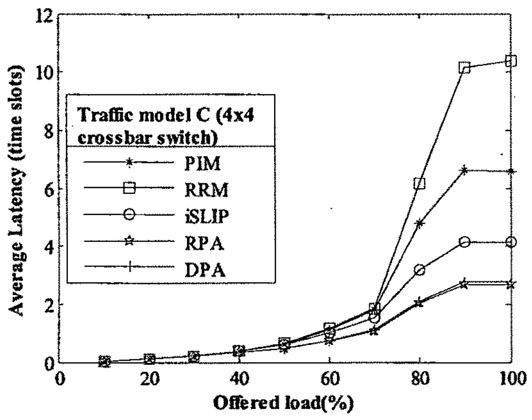


Figure 5.27 Average Latency v/s offered load

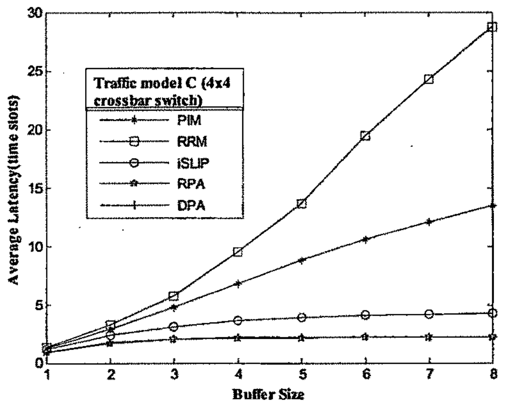


Figure 5.28 Average Latency v/s buffer size

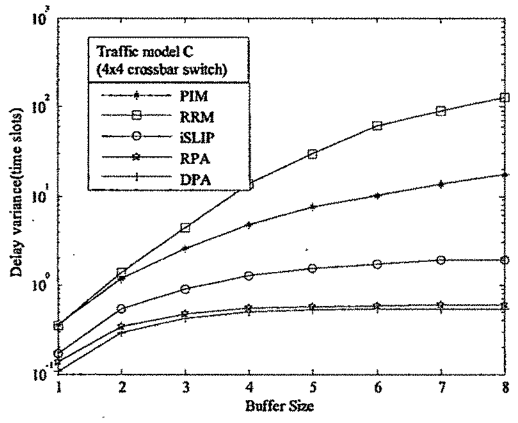


Figure 5.29 Delay variance v/s offered load

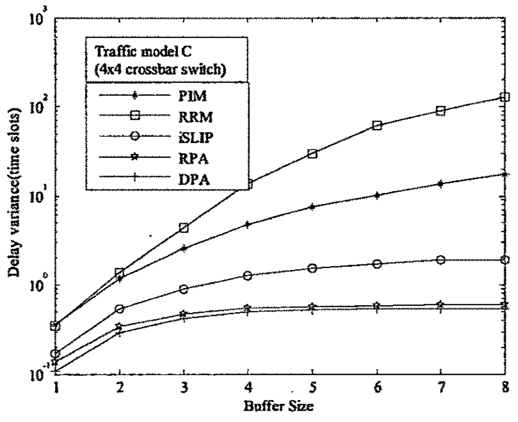


Figure 5.30 Delay variance v/s buffer size

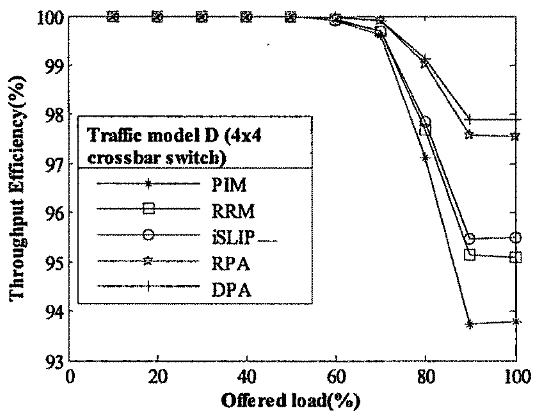


Figure 5.31 Throughput (%) v/s offered load

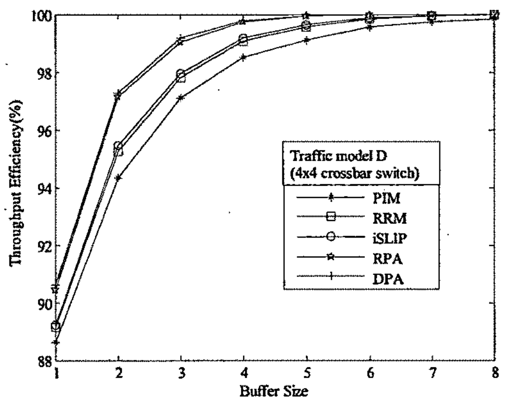


Figure 5.32 Throughput (%) v/s buffer size

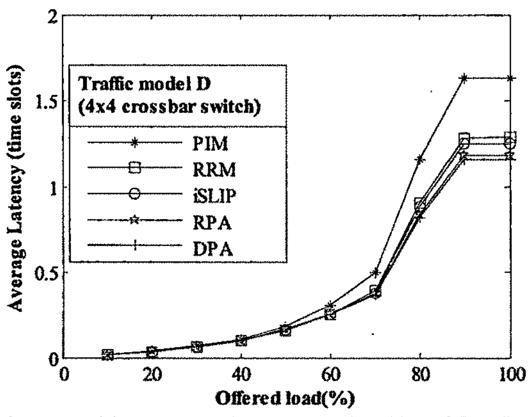


Figure 5.33 Average Latency v/s offered load

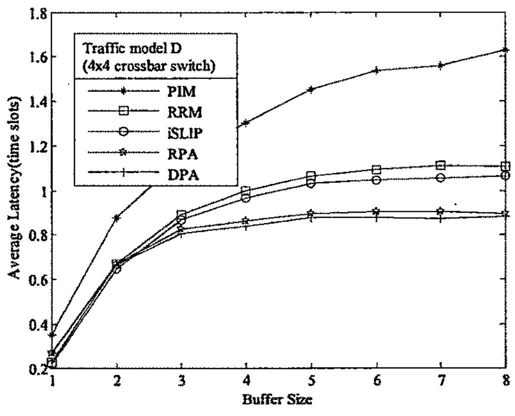


Figure 5.34 Average Latency v/s buffer size

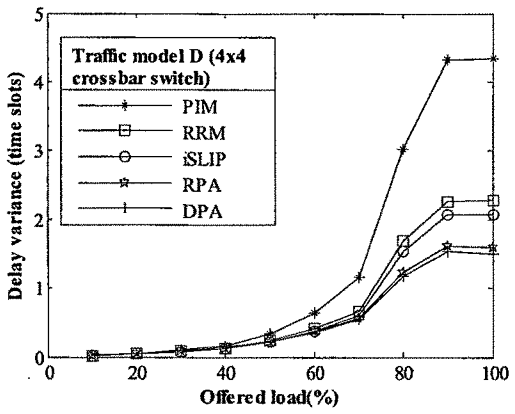


Figure 5.35 Delay variance v/s offered load

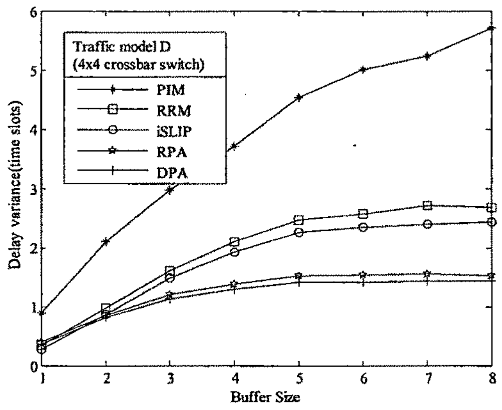


Figure 5.36 Delay variance v/s buffer size

As shown in figure 5.13, Throughput v/s offered load is 3% to 6% higher for RPA and DPA as compared to iSLIP for traffic model –A. In RRM, throughput of RRM is the lowest due to problem in updating the pointers. Grant pointer change in lock-step, like in cell time 1, all point to input 2 and during cell time 2 all point to input 3. This synchronization phenomenon leads to a maximum throughput of 58% for this traffic pattern, at 100% offered load. The overall throughput v/s offered load in traffic model-B is 5% to 8% less than traffic model-A as shown in figure 5.19 due to normally distributed traffic pattern. Traffic model-C resembles to traffic model-A with 10% to 12% more throughput as compared to traffic model – A for all the five scheduling algorithms due to Markov on-off type traffic pattern as shown in figure 5.25. In traffic model-D throughput v/s offered load is higher despite of normal traffic pattern because of Markov on-off type traffic pattern as shown in figure 5.31.

As shown in figure 5.14, throughput increases with increase in buffer size for PIM, iSLIP, RPA and DPA. In RRM, as buffer size increases, repetitive load increases, schedulers move in lock step and throughput (efficiency) in percentage decreases. In traffic model B, 2 % to 3% increase in throughput for PIM, RRM and iSLIP as we increase buffer size from 1 to 8, while in RPA and DPA 4% to 5% increase in throughput as we increase buffer size from 1 to 8. For traffic model-C as shown in figure 5.26 throughput increases 10% to 15%, as we increase buffer size from 1 to 8 for all scheduling algorithm except RRM. In RRM, due to uniform output distribution, scheduler move in lock step and throughput (efficiency) in percentage decreases. For traffic model-D as shown in figure 5.32 throughput increases 10% as we increase buffer size from 1 to 8 for all scheduling algorithm.

As shown in Figure 5.15 average latency is very small for low offered load for all algorithms. But for higher offered load, average latency is in descending order for RRM, PIM, iSLIP, DPA and RPA. In RRM due to synchronization phenomena throughput reduces and hence average latency increases. For traffic model-B as shown in figure 5.21 average latency is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C as shown in figure 5.27 average latency is resembles to traffic model-A. For traffic model-D as shown in figure 5.33 average latency increases with increase in offered load for all algorithms and resembles to traffic model-B. In traffic model-D average latency v/s buffer size is less due to normal and on-off type traffic pattern

As shown in Figure 5.16 in traffic model-A average latency is very small for small buffer size. As buffer size increases from 2 to 8 there is a drastic increase in average latency for RRM and PIM compared to RPA and DPA. For traffic model-B as shown in figure 5.22, due to normally distributed traffic pattern average latency is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C as shown in figure 5.28 average latency is in descending order for RRM, PIM, iSLIP, RPA and DPA and average latency is resembles to traffic model-A. For traffic model-D as shown in figure 5.34 average latency increases with increase in buffer size for all algorithms and descending order for PIM, RRM, iSLIP, RPA and DPA. In traffic model-D average latency v/s buffer size is less due to normal and on-off type traffic pattern.

As shown in Figure 5.17, delay variance is very small for low offered load for all algorithms. But for higher offered load delay variance is high for RRM and PIM and it is in descending order for RRM, PIM, iSLIP, DPA and RPA .For traffic model-B as shown in figure 5.23 delay variance is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C as shown in figure 5.29 delay variance is resembles to traffic model-A. For traffic model-D as shown in figure 5.35 delay variance increases with increase in offered load for all algorithms and resembles to traffic model-B.

As shown in Figure 5.18, delay variance is very small for small buffer size for all algorithms. As buffer size increases from 2 to 8 there is drastic increase in delay variance for RRM and PIM compared to RPA and DPA. For traffic model-B as shown in figure 5.24 delay variance is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C as shown in figure 5.30 delay variance is in descending order for RRM, PIM, iSLIP, RPA and DPA. For traffic model-D as shown in figure 5.36 delay variance increases with increase in buffer size for all algorithms but overall delay variance is less due to normal and on-off type traffic pattern.

5.3.2 8x8 Switch Comparison

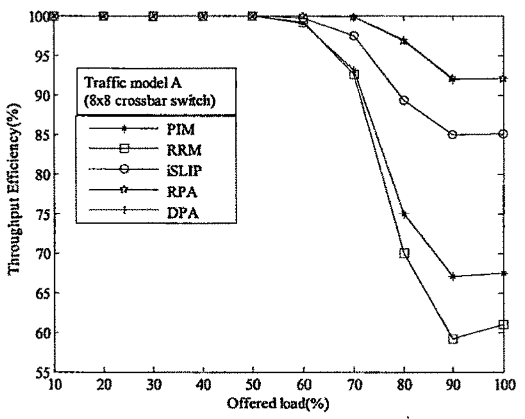


Figure 5.37 Throughput (%) v/s offered load

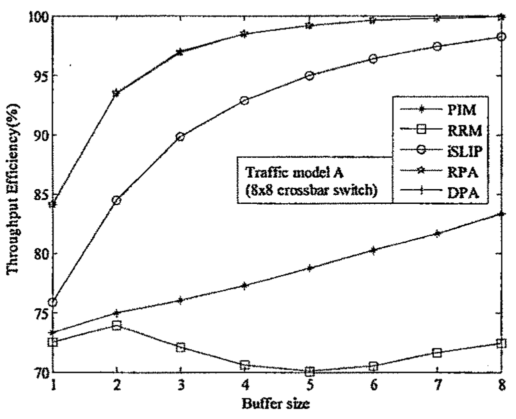


Figure 5.38 Throughput (%) v/s buffer size

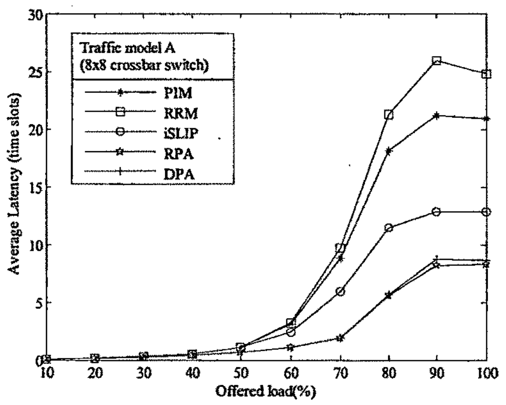


Figure 5.39 Average Latency v/s offered load

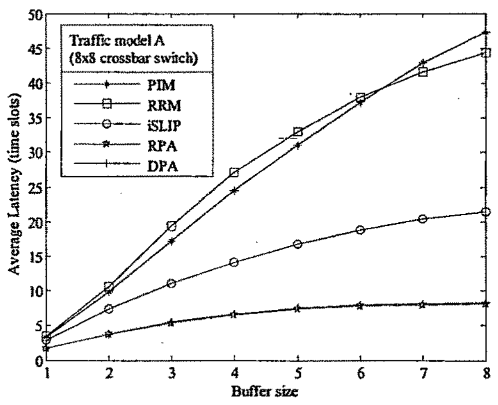


Figure 5.40 Average Latency v/s buffer size

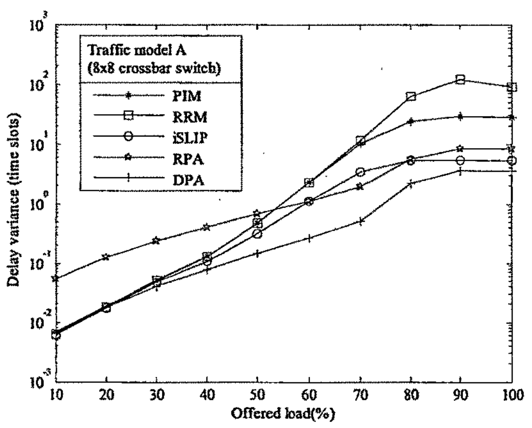


Figure 5.41 Delay variance v/s offered load

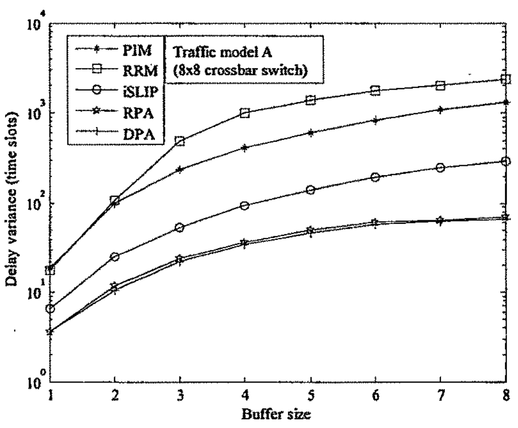


Figure 5.42 Delay variance v/s buffer size

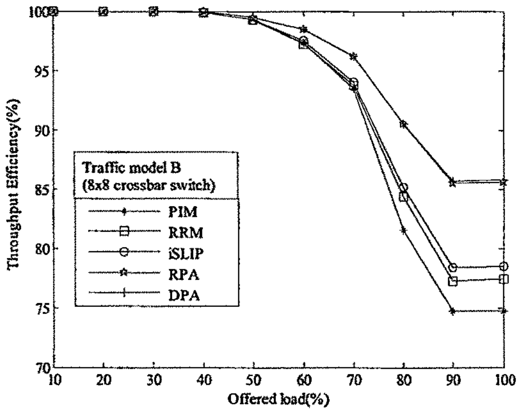


Figure 5.43 Throughput (%) v/s offered load

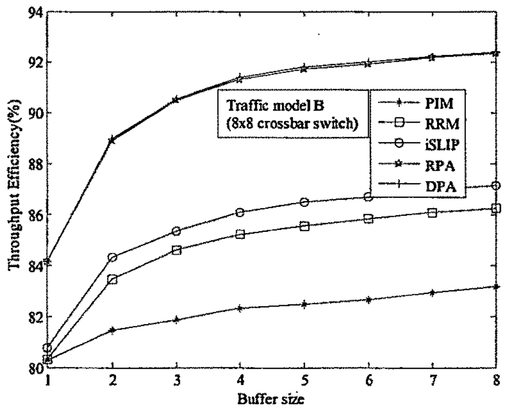


Figure 5.44 Throughput (%) v/s buffer size

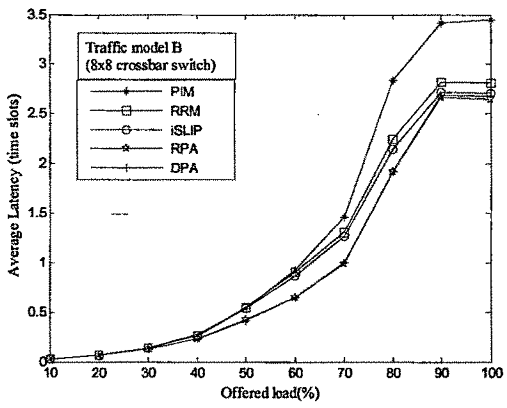


Figure 5.45 Average Latency v/s offered load

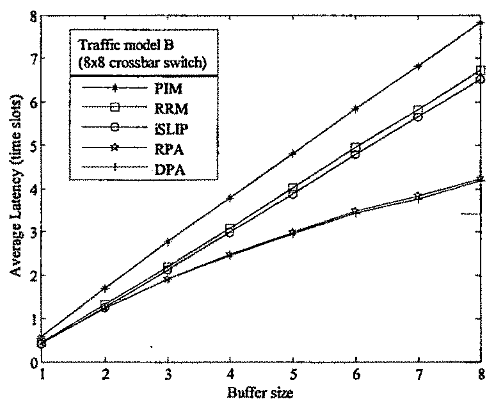


Figure 5.46 Average Latency v/s buffer size

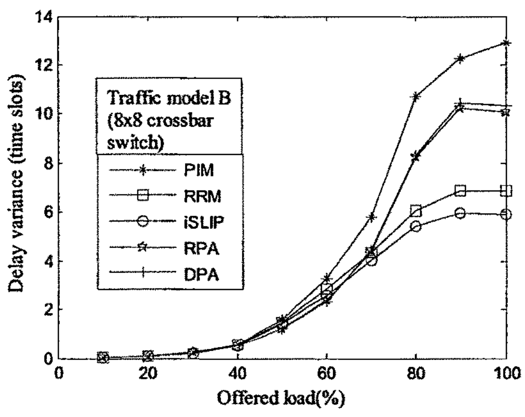


Figure 5.47 Delay variance v/s offered load

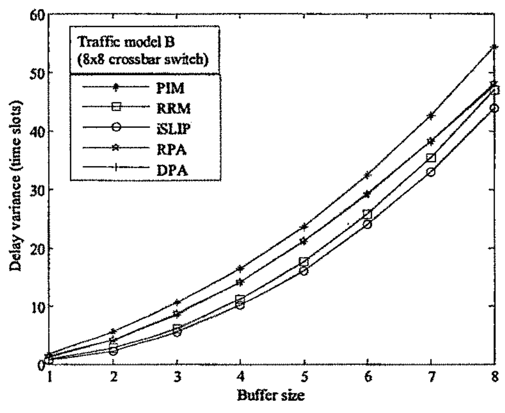


Figure 5.48 Delay variance v/s buffer size

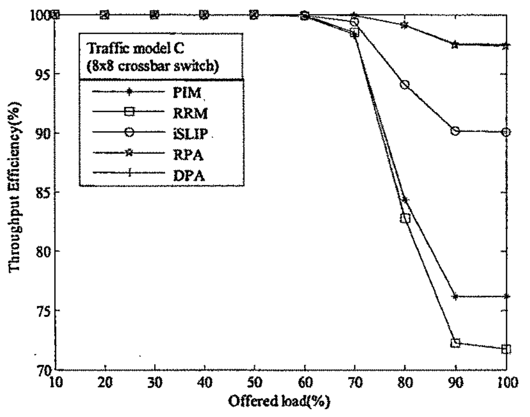


Figure 5.49 Throughput (%) v/s offered load

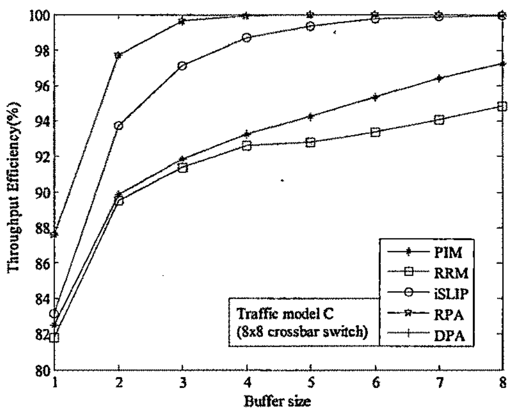


Figure 5.50 Throughput (%) v/s buffer size

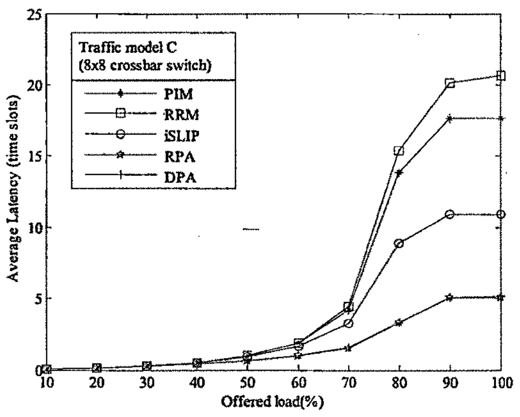


Figure 5.51 Average Latency v/s offered load

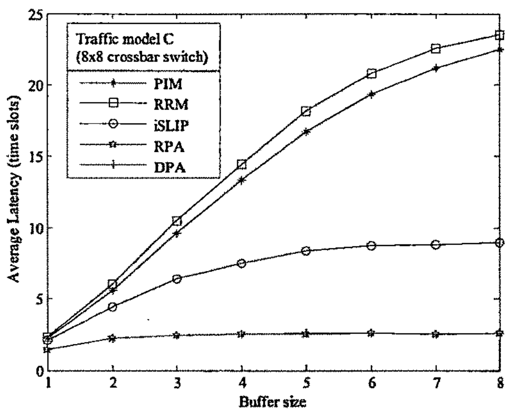


Figure 5.52 Average Latency v/s buffer size

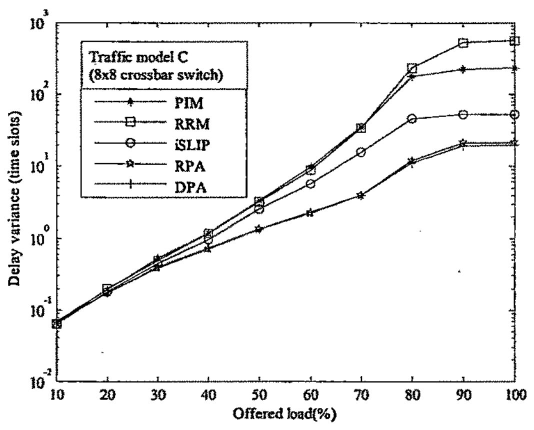


Figure 5.53 Delay variance v/s offered load

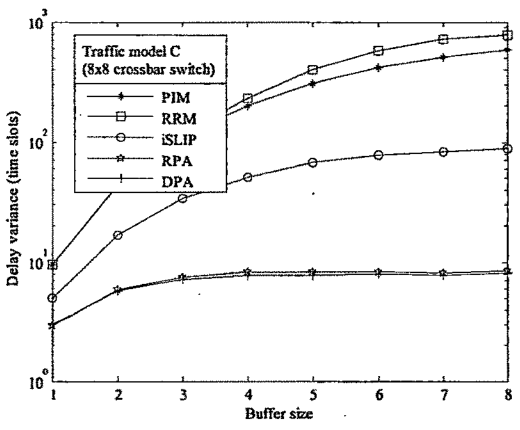


Figure 5.54 Delay variance v/s buffer size

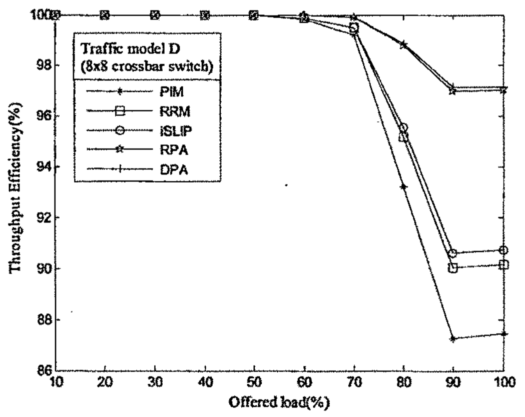


Figure 5.55 Throughput (%) v/s offered load

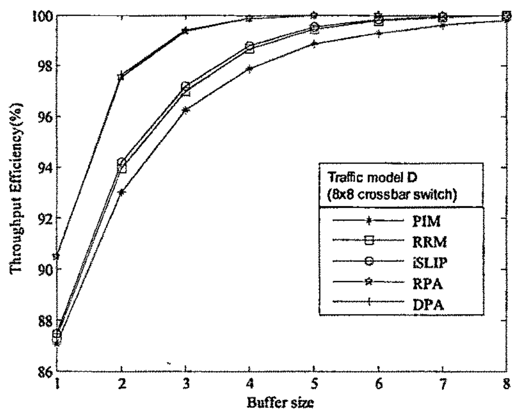


Figure 5.56 Throughput (%) v/s buffer size

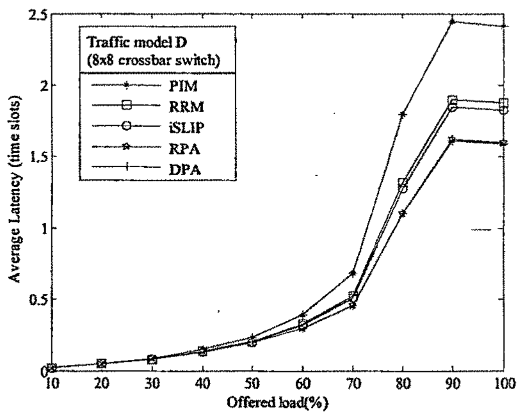


Figure 5.57 Average Latency v/s offered load

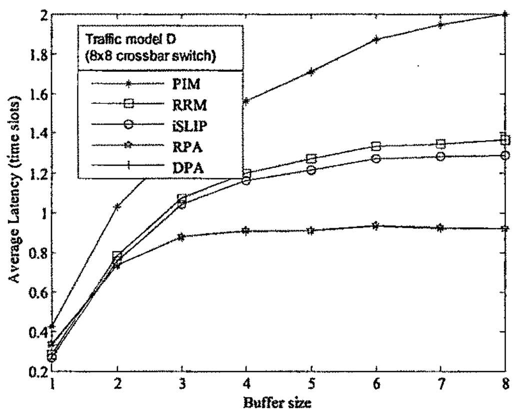


Figure 5.58 Average Latency v/s buffer size

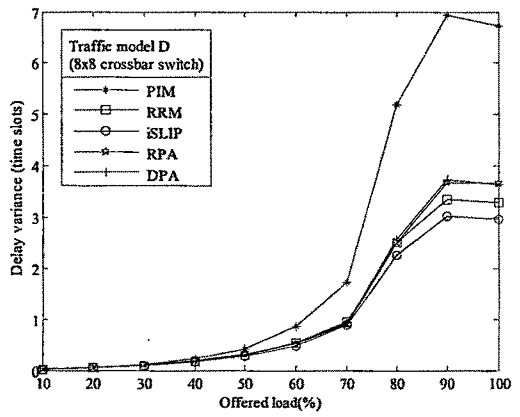


Figure 5.59 Delay variance v/s offered load

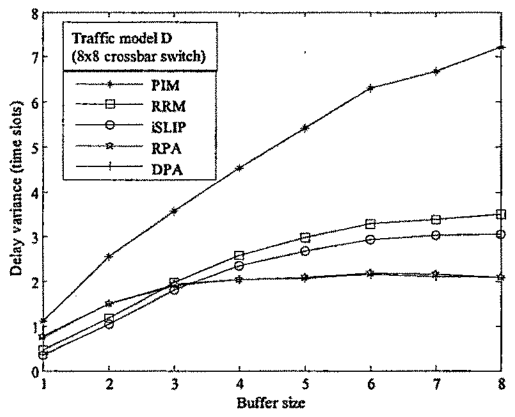


Figure 5.60 Delay variance v/s buffer size

As shown in figure 5.37 throughput v/s offered load is 3% to 8% higher for RPA and DPA as compared to iSLIP for traffic model –A. Throughput of RRM is the lowest due to problem in updating the pointers as discussed earlier. The overall throughput v/s offered load in traffic model-B is 5% to 8% less than traffic model-A as shown in figure 5.43, due to normally distributed traffic pattern. While traffic model-C resembles to traffic model-A, with 5% to 7% more throughput as compared to traffic model –A for all the five scheduling algorithms due to markove on-off type traffic pattern as shown in figure 5.49. In traffic model-D throughput v/s offered load is higher despite of normal traffic pattern because of markove on-off type traffic pattern as shown in figure 5.55.

As shown in figure 5.38, throughput increases, with increase in buffer size for PIM, iSLIP, RPA and DPA. In RRM, as buffer size increases, repetitive load increases, schedulers move in lock step and throughput (efficiency) decreases. As shown in figure 5.44, in traffic model B, 2 % to 4% increase in throughput for PIM, RRM and iSLIP as we increase buffer size from 1 to 8, while in RPA and DPA 4% to 6% increase in throughput as we increase buffer size from 1 to 8. For traffic model-C as shown in figure 5.50 throughput increases 10% to 13% as we increase buffer size from 1 to 8 for all scheduling algorithms. For traffic model-D as shown in figure 5.56 throughput increases 8% to 10% as we increase buffer size from 1 to 8 for all scheduling algorithm.
As shown in Figure 5.39 average latency is very small for low offered load for all algorithms. But for higher offered load average latency is in descending order for RRM, PIM, iSLIP, DPA and RPA. In RRM due to synchronization phenomena throughput reduces and hence average latency increases. For traffic model-B as shown in figure 5.45 average latency is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C, as shown in figure 5.51 average latency is resembles to traffic model-A. For traffic model-D, as shown in figure 5.57 average latency increases with increase in offered load for all algorithms and resembles to traffic model-B. In traffic model-D average latency v/s buffer size is less due to normal and on-off type traffic pattern.

As shown in Figure 5.40 in traffic model-A average latency is very small for small buffer size. As buffer size increases from 2 to 8 there is drastic increase in average latency for RRM and PIM compared to RPA and DPA. For traffic model-B as shown in figure 5.46, due to normally distributed traffic pattern average latency is in descending order for PIM, RRM, iSLIP, RPA and DPA. For traffic model-C as shown in figure 5.52 average latency is in descending order for RRM, PIM, iSLIP, RPA and DPA and average latency resembles to traffic model-A. For traffic model-D as shown in figure 5.58 average latency increases with increase in buffer size for all algorithms and descending order for PIM, RRM, iSLIP, RPA and DPA. In traffic model-D average latency v/s buffer size is less due to normal and on-off type traffic pattern.

As shown in Figure 5.41, delay variance is very small for low offered load for all algorithms. But for higher offered load delay variance is high for RRM and PIM and it is in descending order for RRM, PIM, RPA , iSLIP, and DPA. For traffic model-B, as shown in figure 5.47, delay variance is in descending order for PIM, DPA, RPA, RRM and iSLIP. For traffic model-C as shown in figure 5.53 delay variance resembles to traffic model-A. For traffic model-D as shown in figure 5.59, delay variance increases with increase in offered load for all algorithms and resembles to traffic model-B.

As shown in Figure 5.42, delay variance is very small for small buffer size for all algorithms. As buffer size increases from 2 to 8 there is drastic increase in delay variance for RRM and PIM compared to RPA and DPA. For traffic model-B as shown in figure 5.48, delay variance is in descending order for PIM, DPA , RPA, RRM and iSLIP. For traffic model-C as shown in figure 5.54 delay variance is in descending order for RRM, PIM, iSLIP, RPA and DPA. For traffic model-D as shown in figure 5.60 delay variance increases with increase in buffer size for all algorithms but overall delay variance is less due to normal and on-off type traffic pattern.

5.3.3 16x16 Switch Comparison

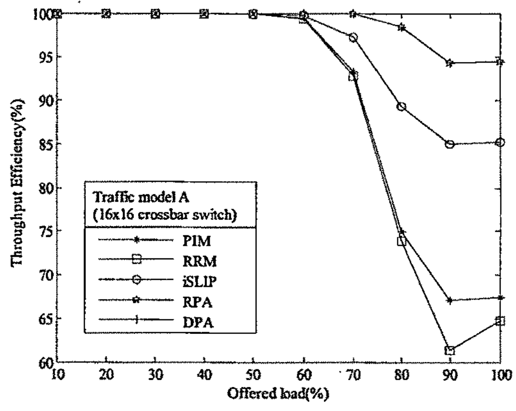


Figure 5.61 Throughput (%) v/s offered load

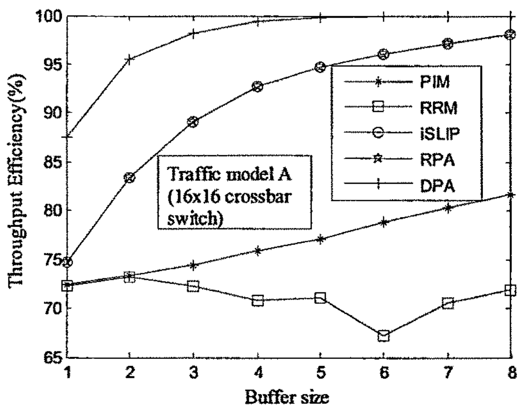


Figure 5.62 Throughput (%) v/s buffer size

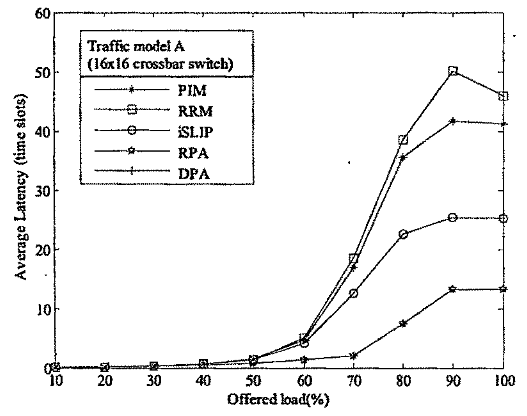


Figure 5.63 Average Latency v/s offered load

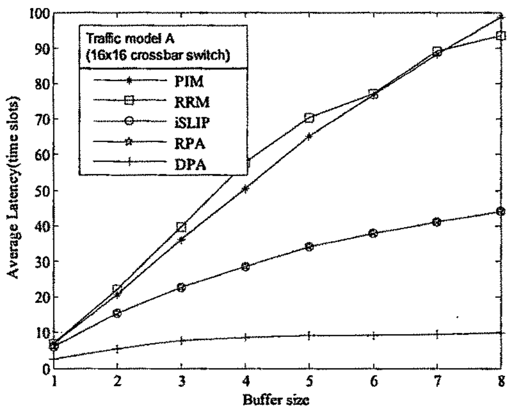


Figure 5.64 Average Latency v/s buffer size

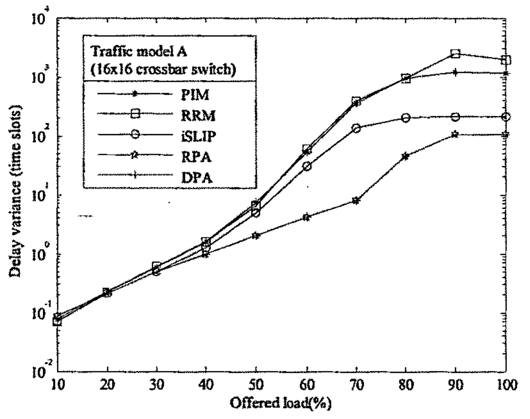


Figure 5.65 Delay variance v/s offered load

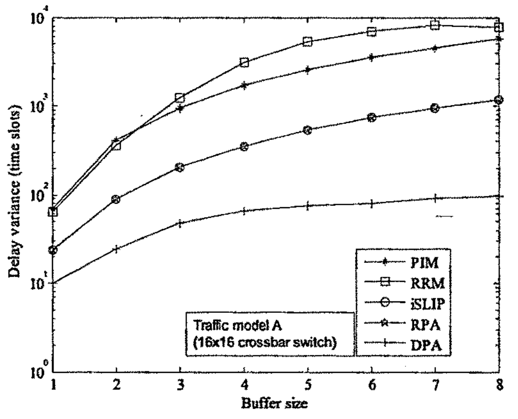


Figure 5.66 Delay variance v/s buffer size

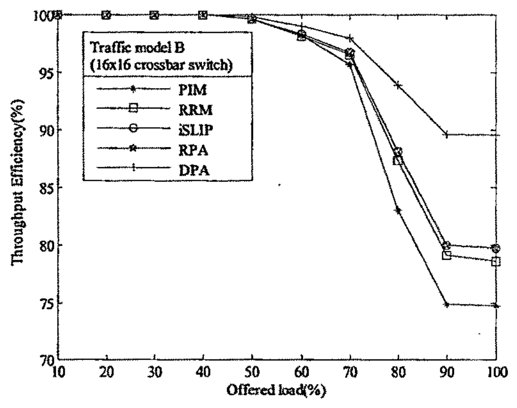
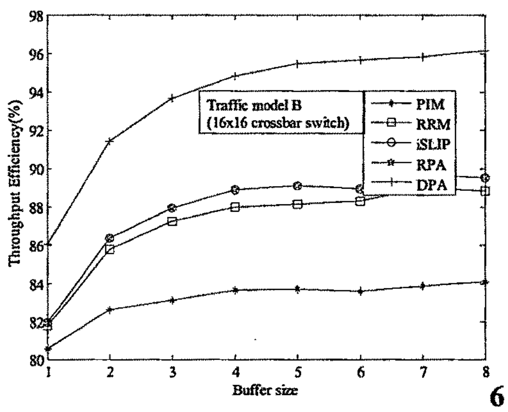


Figure 5.67 Throughput (%) v/s offered load



6

Figure 5.68 Throughput (%) v/s buffer size

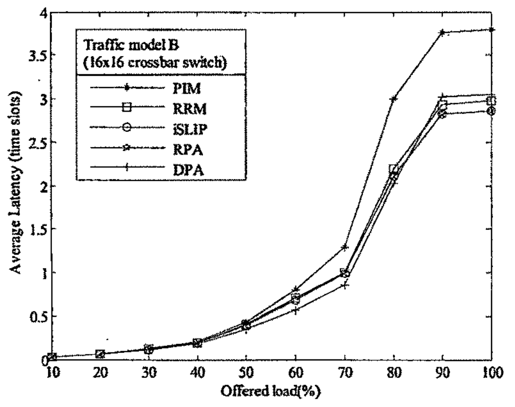


Figure 5.69 Average Latency v/s offered load

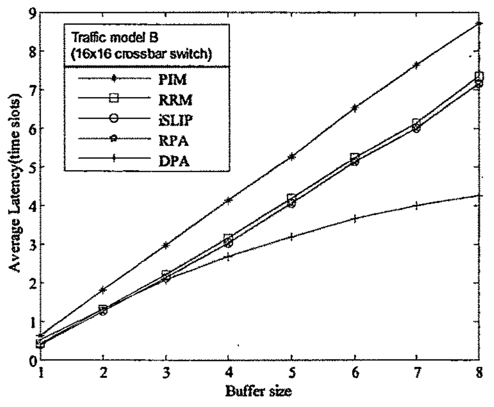


Figure 5.70 Average Latency v/s buffer size

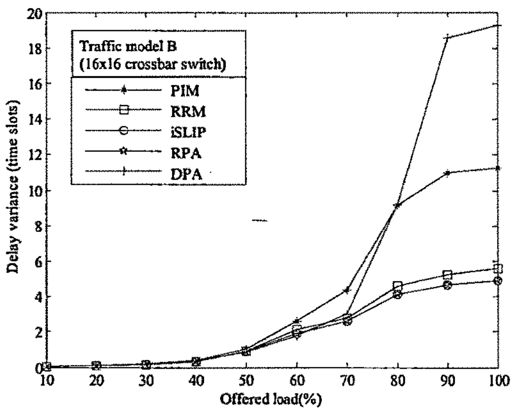


Figure 5.71 Delay variance v/s offered load

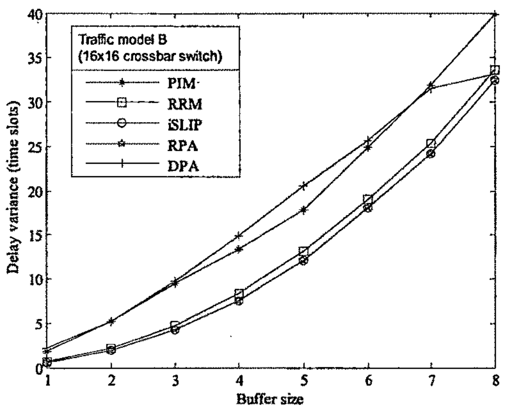


Figure 5.72 Delay variance v/s buffer size

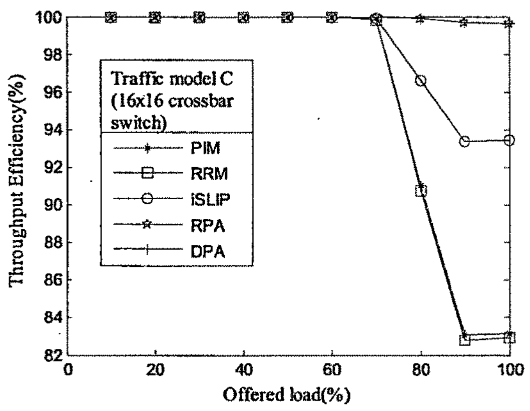


Figure 5.73 Throughput (%) v/s offered load

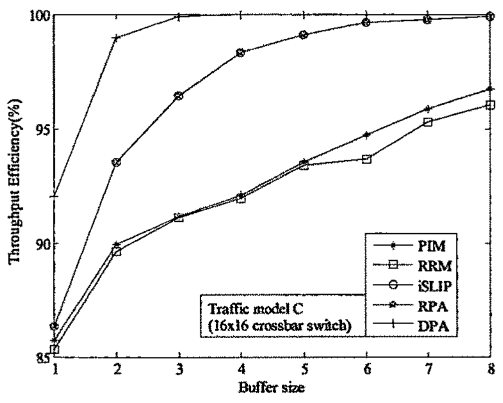


Figure 5.74 Throughput (%) v/s buffer size

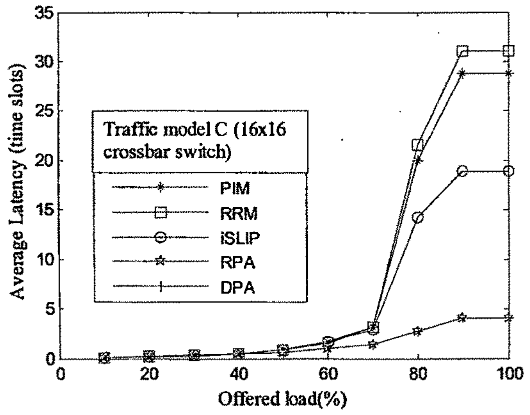


Figure 5.75 Average Latency v/s offered load

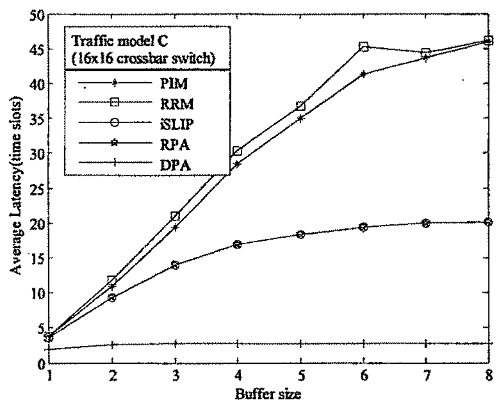


Figure 5.76 Average Latency v/s buffer size

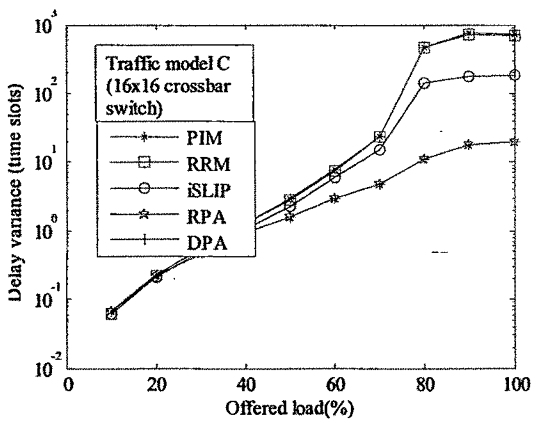


Figure 5.77 Delay variance v/s offered load

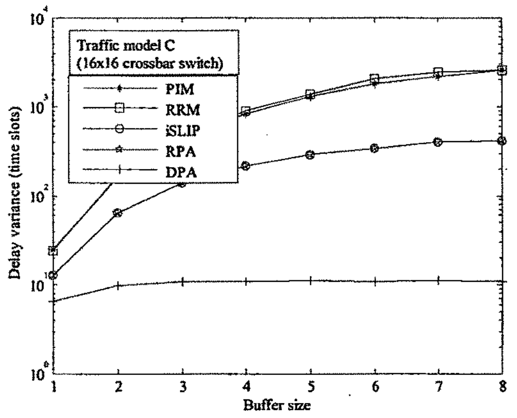


Figure 5.78 Delay variance v/s buffer size

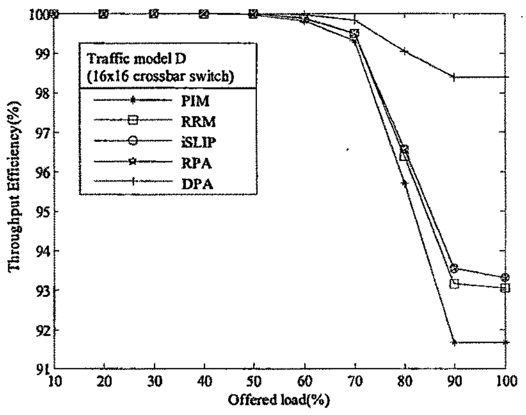


Figure 5.79 Throughput (%) v/s offered load

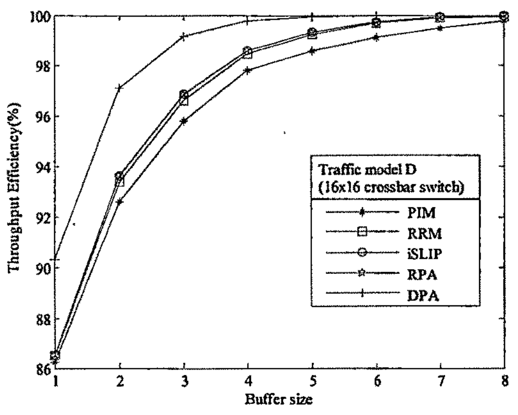


Figure 5.80 Throughput (%) v/s buffer size

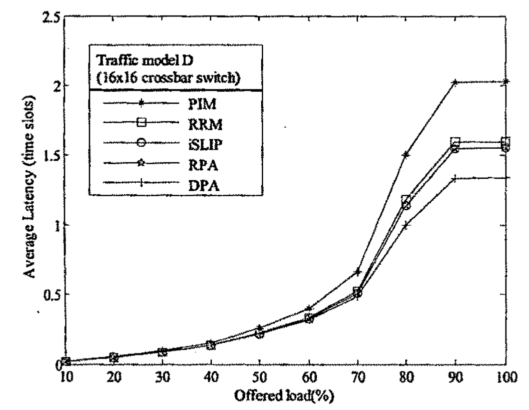


Figure 5.81 Average Latency v/s offered load

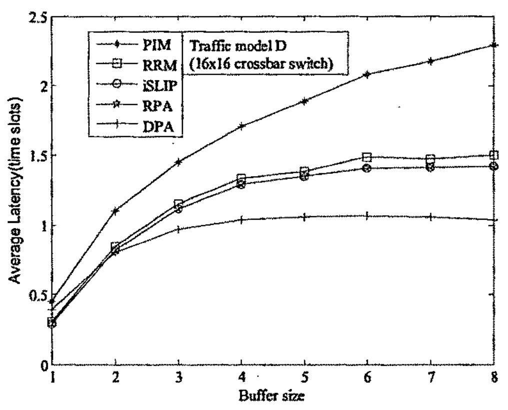


Figure 5.82 Average Latency v/s buffer size

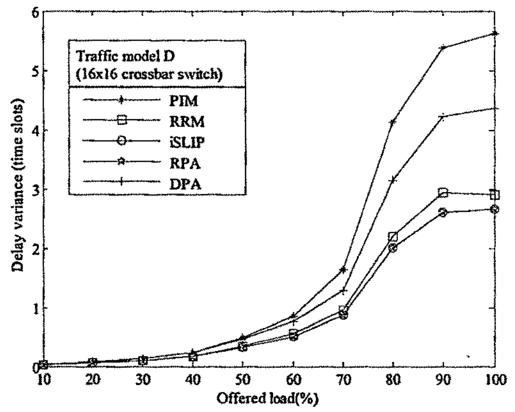


Figure 5.83 Delay variance v/s offered load

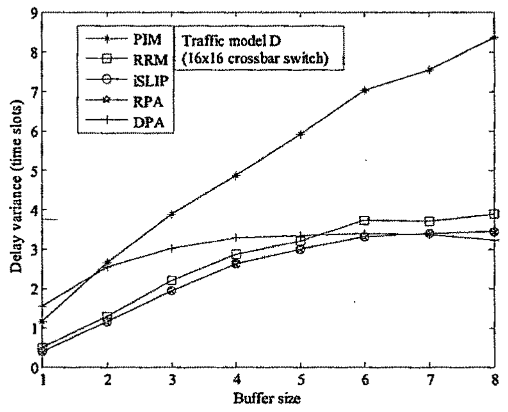


Figure 5.84 Delay variance v/s buffer size

The analysis of 16x16 switch results resembles that of 8x8 switch.

5.3.4 32x32 Switch Comparison

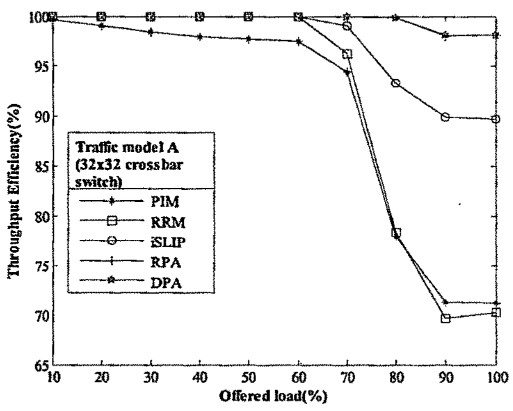


Figure 5.85 Throughput (%) v/s offered load

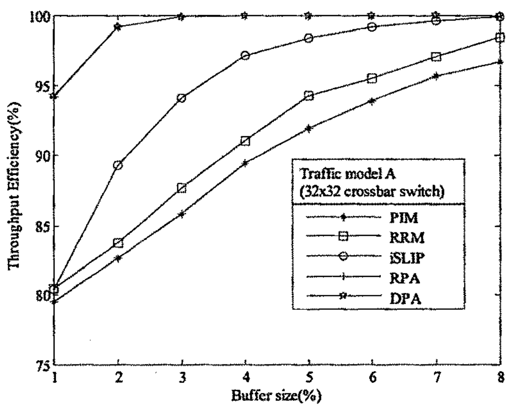


Figure 5.86 Throughput (%) v/s buffer size

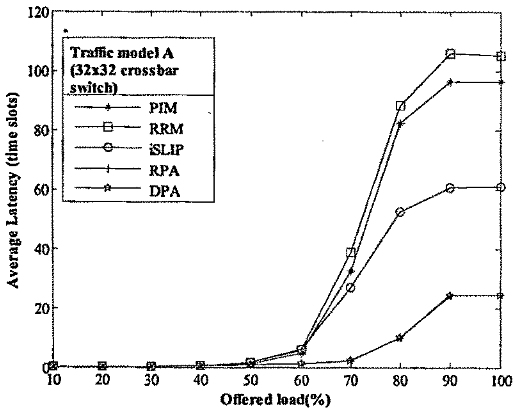


Figure 5.87 Average Latency v/s offered load

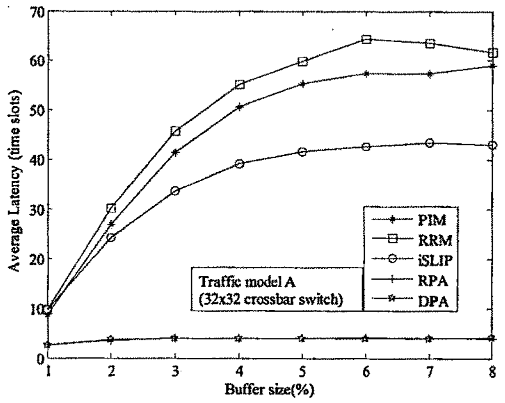


Figure 5.88 Average Latency v/s buffer size

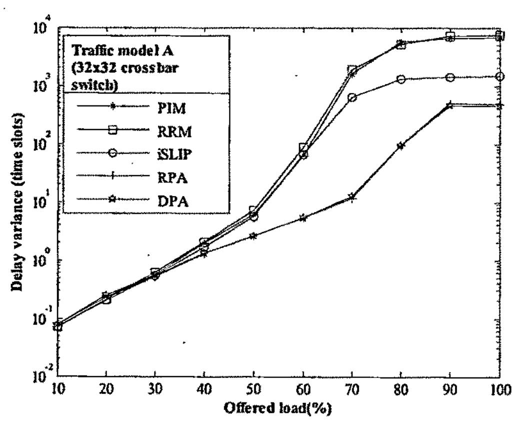


Figure 5.89 Delay variance v/s offered load

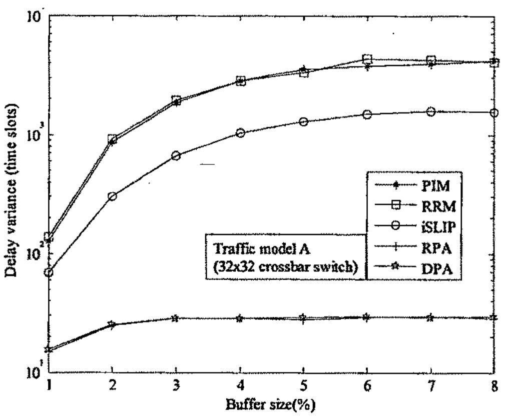


Figure 5.90 Delay variance v/s buffer size

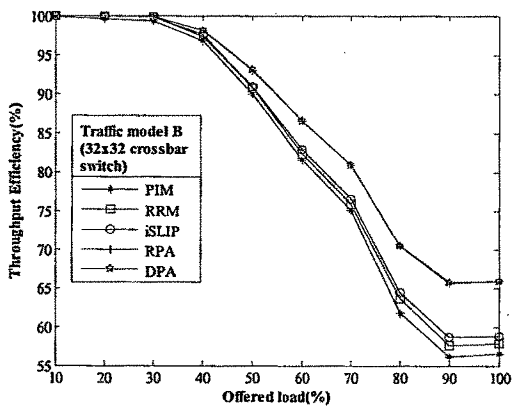


Figure 5.91 Throughput (%) v/s offered load

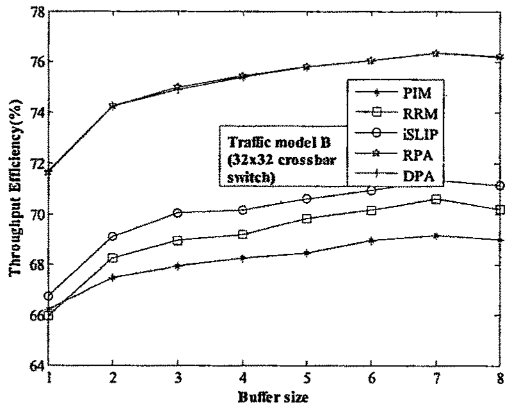


Figure 5.92 Throughput (%) v/s buffer size

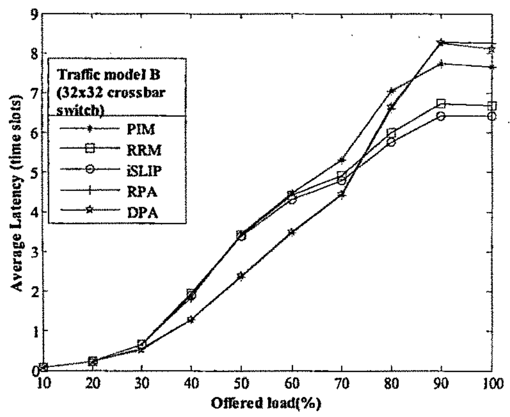


Figure 5.93 Average Latency v/s offered load

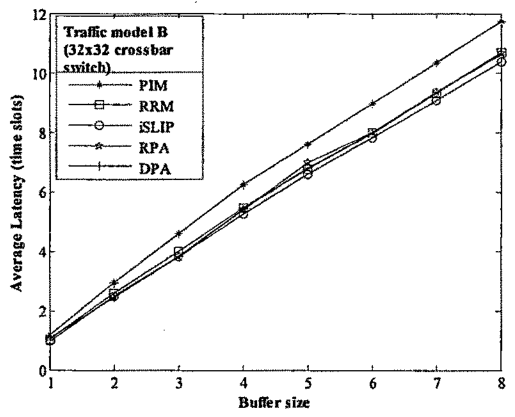


Figure 5.94 Average Latency v/s buffer size

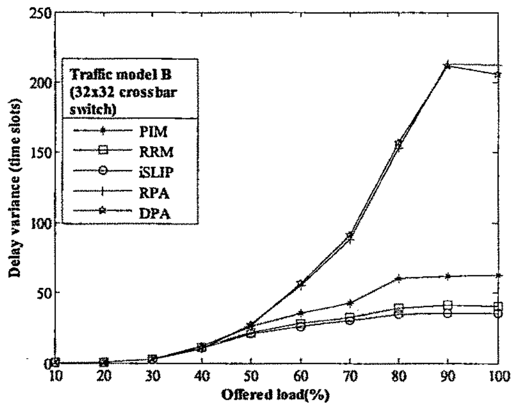


Figure 5.95 Delay variance v/s offered load

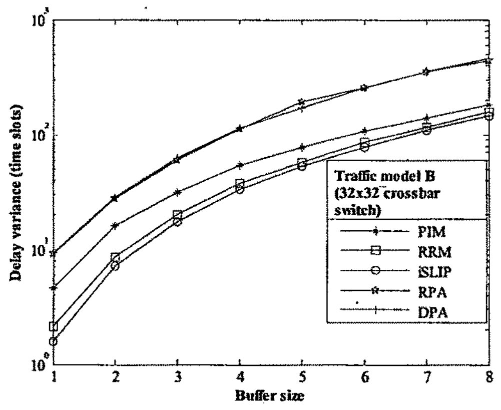


Figure 5.96 Delay variance v/s buffer size

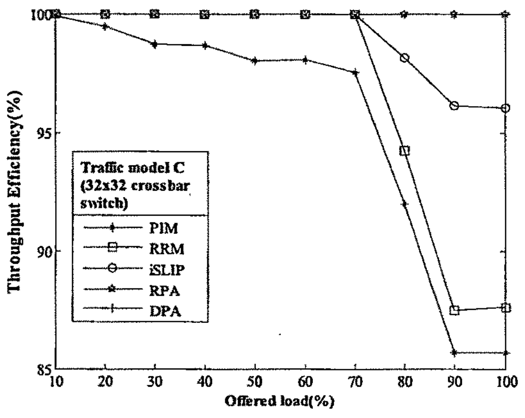


Figure 5.97 Throughput (%) v/s offered load

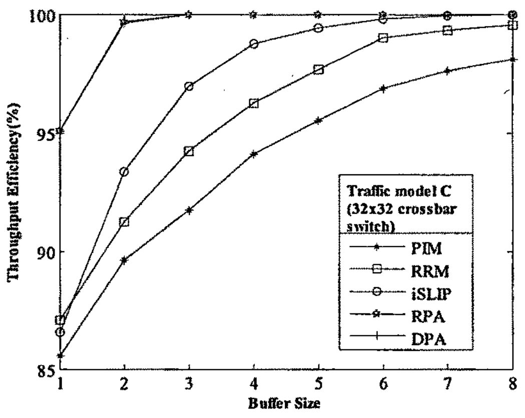


Figure 5.98 Throughput (%) v/s buffer size

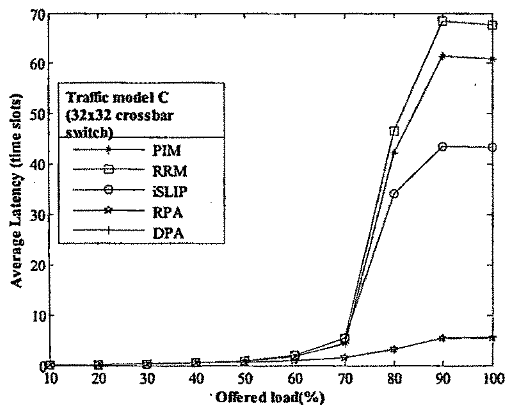


Figure 5.99 Average Latency v/s offered load

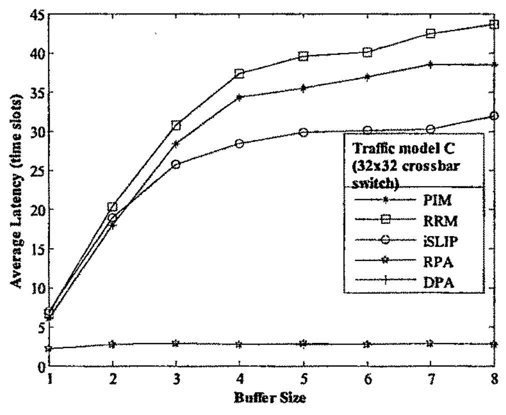


Figure 5.100 Average Latency v/s buffer size

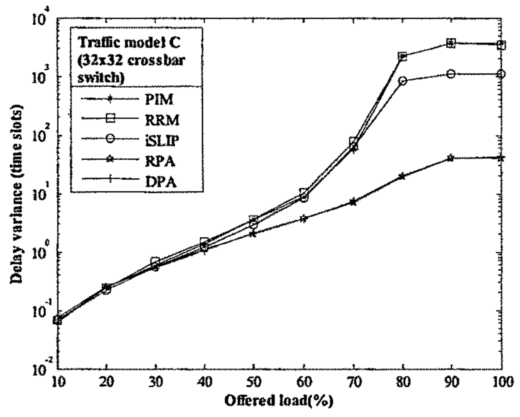


Figure 5.101 Delay variance v/s offered load

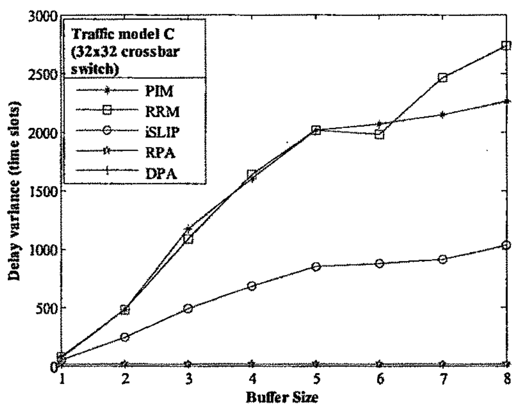


Figure 5.102 Delay variance v/s buffer size

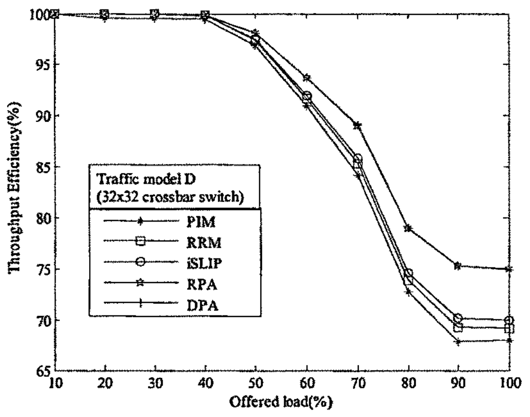


Figure 5.103 Throughput (%) v/s offered load

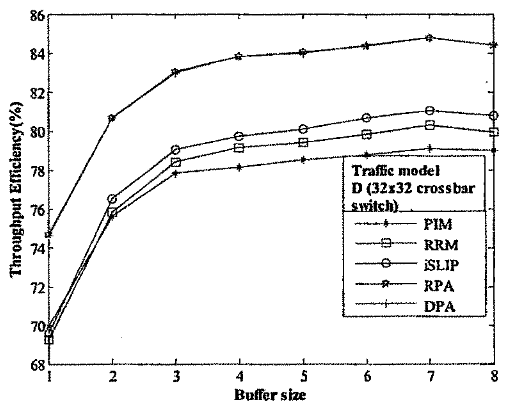
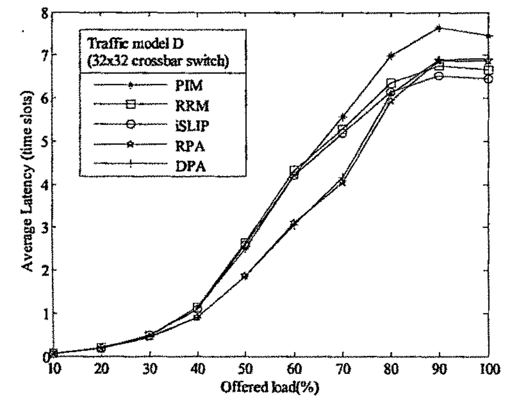


Figure 5.104 Throughput (%) v/s buffer size



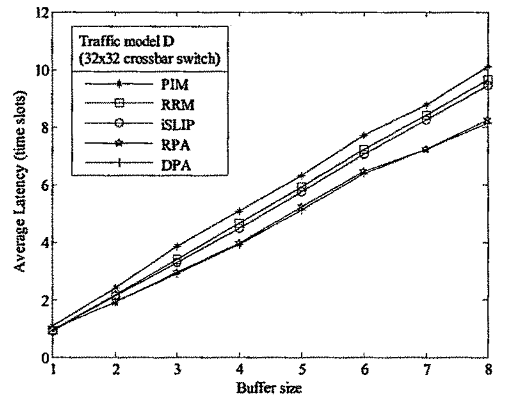
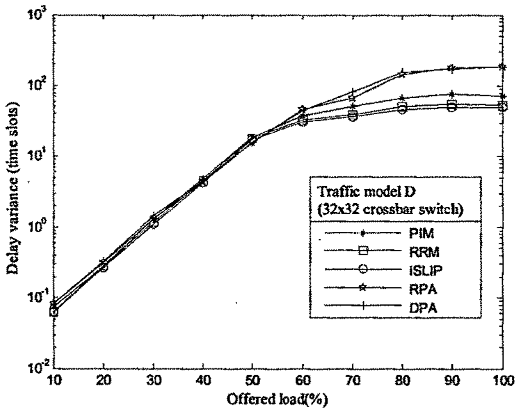


Figure 5.105 Average Latency v/s offered load **Figure 5.106 Average Latency v/s buffer size**



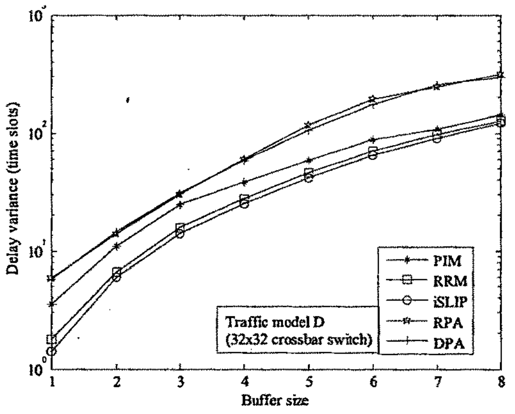


Figure 5.107 Delay variance v/s offered load **Figure 5.108 Delay variance v/s buffer size**

The analysis of 32x32 switch results resembles that of 16x16 and 8x8 switches.

5.4 VLSI IMPLEMENTATION OF THE SCHEDULING ALGORITHMS

The basic architecture of input queue NxN cross bar switch is given in Figure 5.109. There are N Input VOQ buffers (input_port) blocks, one scheduler block and one switch fabric block in the NxN ATM cross bar switch. IP packets have to be fragmented into ATM cells before being input to the switch. The input lines to the switch are N data lines (each of 8 bit size), N frame start inputs, one clock input, and a reset input. The output lines of the switch are N data output lines(each of 8 bit size), N data valid lines, N output frame pulse lines, one clock output, and N outputs that indicate the origin of the data coming to each data output port. The N data inputs are each 8 bits wide, and carry 53 byte Asynchronous Transfer Mode (ATM) packets. We use the rising edge of the clock signal to input and output the data. One clock, called *s_clk* which has a period equal to a packet time is internally generated within the input port modules and used in some of the scheduling algorithms. Similarly N *used_word* signals are generated from each input_port module, which give status of the VOQ buffers and used in proposed scheduling algorithms like m-DPA and DSA discussed in chapter 6 of the thesis. Port_request signal of N bits from all the N input VOQ buffer module, sheduler_clk and N^2 *used_word* signals each of 8 bits are given to the scheduler depending on the requirement of scheduling algorithm. Scheduling algorithm generates N grant signals each of N bits. These grant signals are given to respective input_port module to release the data. This data is outputted through switch fabric. The frame start inputs are one clock cycle wide signals indicating the start of packets. The *reset* input of the switch reset all the counters used in the design and initialize them to their starting values. The output frame start signals indicate the beginning of outgoing packets for their corresponding data lines. The data_valid output lines indicate whether the data present at the corresponding output of the switch is valid for sampling. A src_no signal each of $\log_2 N$ bits at each output port indicates from, which input port the data is originated. This signal can later be used for classifying, outputting and reassembling the data.

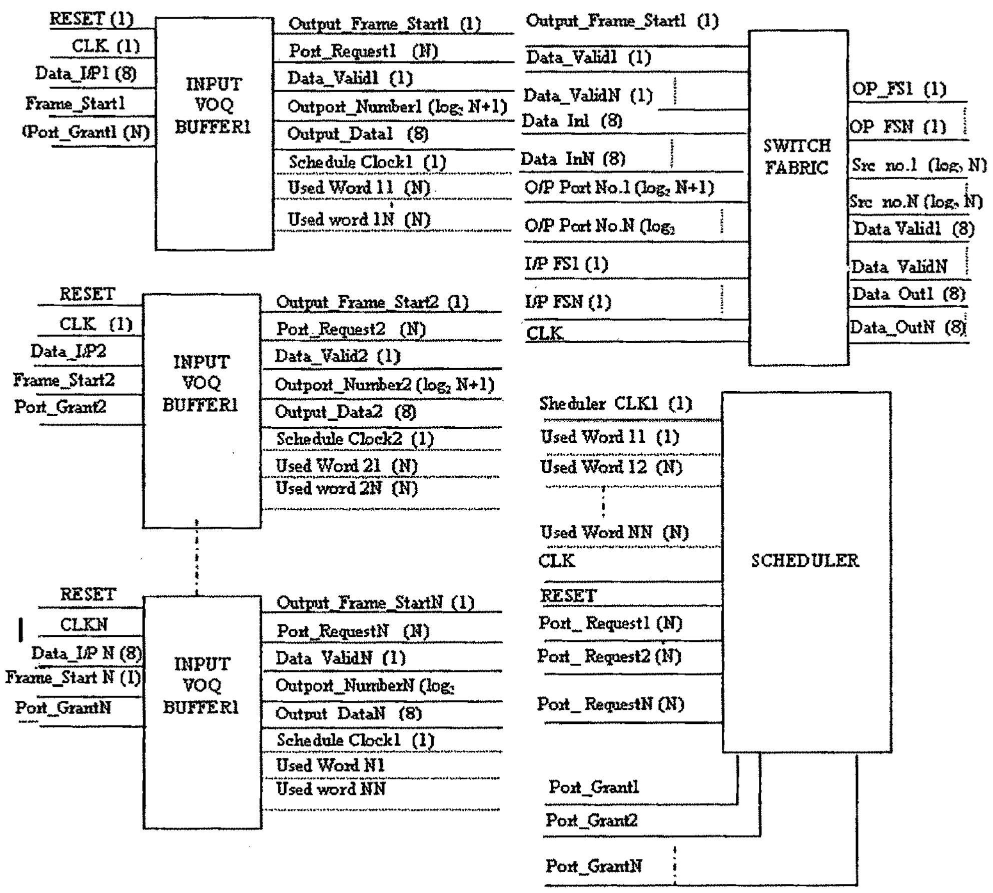


Figure 5.109 Basic Architecture of input queue NxN cross bar switch

5.4.1 Input VOQ Queuing Device (Input_Port)

Input buffer module is responsible for handling, storing and processing the arriving ATM packets. Each data byte arriving at the input_port module is first delayed by few clock cycles, so that during that delay, it will retrieve new VCI value as well as destination output port number from the Look Up Table (LUT) and then send it as an input of the FIFO.

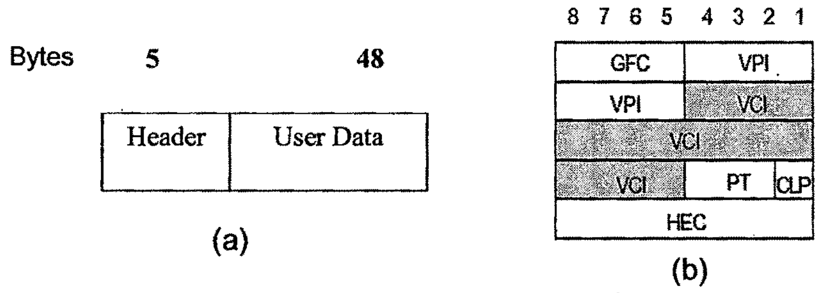


Figure 5.110 (a) An ATM cell. (b) ATM cell header detail

Figure 5.110 shows an ATM cell with its header and payload bytes. The second, third and fourth bytes of the packet having Virtual Circuit Identifier (VCI) information are written into VCI registers as well as the buffer. After the first four bytes of a packet are read, the input port extracts the address information (VCI bits) from the header of the arriving ATM packet and sends it to a LUT module. The LUT returns the new VCI together with the destination output port number for that packet. The input port then sends a request for that specific output port to the scheduler, and awaits a grant.

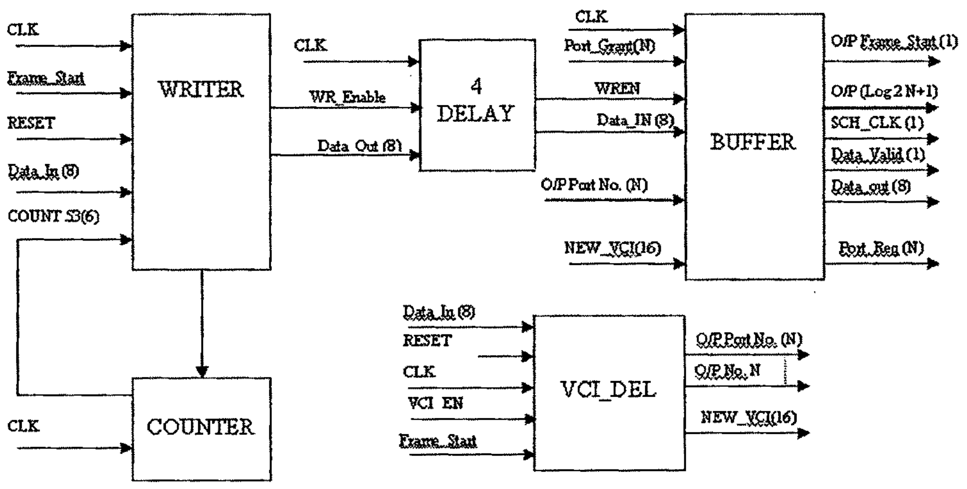


Figure 5.111 Input_port internal architecture

Once a grant is released for a certain packet, the data bytes are de-queued first in first out basis from the FIFO of the input buffer. After the entire packet is sent, the same process is repeated for the next packet. As soon as a grant for an output port is issued, the input port number is sent to the crossbar fabric so that the output port receiving the data knows the origin of the packet. Figure 5.111 shows a detailed schematic of the input_port module.

The VOQ buffer holds up to 212N [53 (byte) x N (outputs) x 4 (buffer size for each output)] one-byte words as shown in figure 5.112. The choice buffer size is a trade off between the switch speed and the loss rate. The larger the buffer, smaller is the probability of buffer overflow and the loss rate. On the other hand, the queuing delay increases as the buffer size grows, as discuss in simulation. A large queuing delay reduces the switching speed and results in a low Quality of Service (QoS) in the network. There are two counters used in the implementation. The first counter is 6 bits wide and counts the number of bytes that enter the input_port module. This counter is set again, once the whole packet is read. The second counter is used for delaying the data bytes going to the buffer as its input.

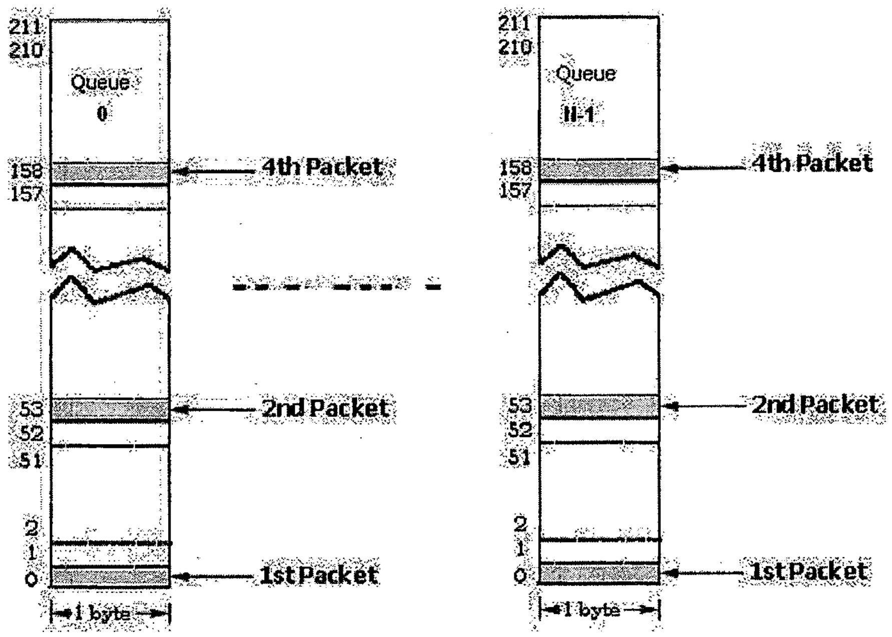


Figure 5.112 The VOQ buffer in each input_port module.

The Look Up Table is implemented in a ROM. ROM can be initialized with an arbitrary set of data in a filename.mif file. The LUT searches through the ROM rows, until it finds a match between the input VCI bits in the ROM and the *input_vci* input to the LUT. If the match exists on row m of the ROM, the output VCI bits and the output port number bits in row m are displayed on *output_vci* and *output_port_no* outputs of the LUT component, respectively. The look-up table in all the input port modules of these switches is initialized with the same values for simplicity reasons. These values are shown in Table 5.1.

Table 5.1: The Look-up table Detail of the switch.

Input vci	Output vci	Output port no.
3080	E965	1
7747	F6A9	1
2E1E	59E0	2
13E3	B10A	3
2ABA	3BED	0
24D4	FA25	2
2171	0106	3
6838	FA65	0

Initially for four bytes, enable signal is set to logic one using counter, so these bytes are written into the VCI registers as well as the buffer itself. These bytes contain the VCI information needed for routing the packet through the switch. As soon as all the bytes of the packet are written and counter reaches at terminal count, it resets and awaits the arrival of a new packet. So the data is being written into the FIFO according to its destination output port number with updated vci value, input port sends the request to the scheduler. After receiving the request from the input port, scheduler sends grants to the one of the queue intended for the same output port. After receiving grant from the scheduler, input port starts reading packets from the granted FIFO's queue and sends it to the input of the switching fabric.

5.4.2 Cross Point Switch (Switch Fabric)

The switch fabric module physically connects an input port to its destined output port, based on the output port number generated by the input port. The outputs of input port modules are connected to the inputs of switch_fabric. The outputs of switch_fabric are connected to the output ports of the switch.

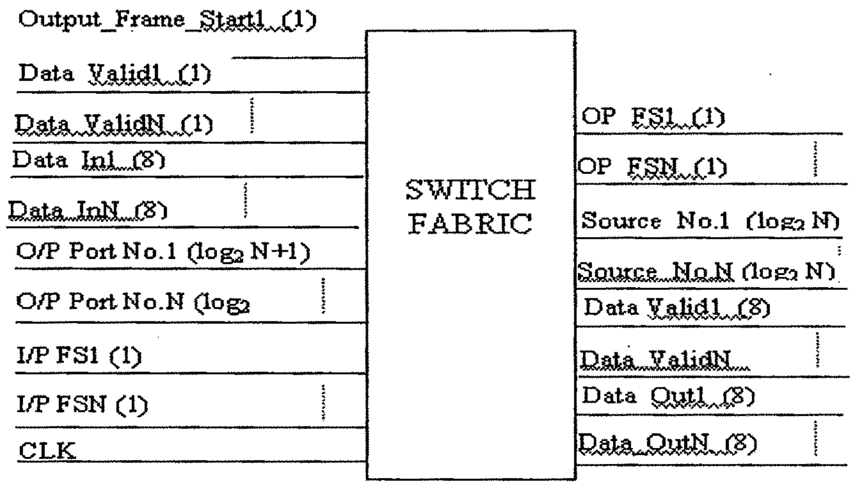


Figure 5.113 General NxN Switch fabric module.

As shown in figure 5.113, Input signals to the switch fabric module are: *clk*, *data_valid* (N x1 i.e. N signals of 1 bit), data_in (Nx8), op_port_no(N x (log_2N +1)), frame start (N x1). Output signals from the switch fabric are: op_fs (Nx1), source_no (Nx (log_2N)), *data_valid* (N x1), and data_out (Nx8). Input signal data_valid indicates that there is a valid 8 bit data at the respective data_in input. There are N input signals *op_port_no* of size (log_2N +1). In each signal log_2N bit indicates destination number to switch. One more bit is required to validate the data like if "000" 00 is the destination number and valid data, but "100" means invalid data.

5.4.3 Scheduler

Depending on various scheduling algorithms, scheduler generates the grants depending on the request generated by various inputs and scheduling policy.

5.5 VLSI IMPLEMENTATION OF RRM

As shown in figure 5.114, RRM scheduler block consists of 5 sub blocks. Clkipdrrm block has two clock inputs: schedule clock (sc0) and general clock (clk). At every packet arrival time, it generates four output clocks, (ctmp0, ctmp1, ctmp2, ctmp3) sequentially in synchronism with main system clock (clk), once sc0 is detected as shown in figure 5.119. This output clocks (ctmp0, ctmp1, ctmp2, ctmp3) are given to four other individual blocks as shown in figure 5.114. Input to the block Port_req_gen are port request signals generated by individual input ports. We adjust the request signals in terms of output in such a way that for each individual output, there is a 4 bit signal which indicates the request from 4 different input ports as shown in figure 5.115. Grt_rrm block decides grant of individual outputs. There are N counters required to generate grant for NxN switch. We have implemented 4x4 and 8x8 crossbar switches so we need 4 and 8 counters respectively. When there is a conflict, i.e. more than one input port wants to transfer data to same output, then based on the grant pointer, grant will be given to particular input port as illustrated in figure 5.1. Figure 5.116 indicates the waveforms of GRT_RRM blocks for 4x4 switch. Gr_ipx indicates the request of the inputs at output x. Grant counter (gnt_cntx) indicates the grant pointers at output x, and gr_opx displays the output of x. As shown in figure 5.116 at 36.0 ns, we set the following inputs; gr_ip3= 0110, gr_ip2= 1011, gr_ip1= 0000 and gr_ip0=0111. Initially all grant counters (gnt_cntx) are 00.In case of output3 (gr_ip3=0110), there are two requests from input1 and input2 respectively. Grant counter (gnt_cnt=00) points to 0 so grant will be given to input1 and gr_op3=0010 and grant counter points to the next input (gnt_cnt3=10). Similarly, depending on grant input and initial value of grant counter, grant output are set like this; gr_op3=0010, gr_op2=0001, gr_op1=0000, gr_op0=0001 and the grant counters are set to: gr_cnt3=10, gr_cnt2=01, gr_cnt1=00, gr_cnt0=01 (Modulo N increment to grant output).Same process is repeated after next clock (at 200.0 ns.), with updated grant counter.

Input to the block Port_grt_gen are grant signals generated by Grt_rrm block. We adjust the Grant signals in terms of input in such a way that for each individual input there is a 4-bit signal which indicates the request from 4 different output ports as shown in figure 5.117.

Acpt_rrm block decides acceptance of individual inputs. There are N counters required to generate acceptance for NxN switch. We implement 4x4 and 8x8 crossbar switches so we need 4 and 8 counters respectively. When there is a conflict, i.e. more than one output port give grant to same input, then based on the accept pointer, accept signal will be given to particular output port as shown in figure 5.1.

As shown in figure 5.118 at 35.0 ns, we set the following accept inputs; acpt_ip3= 0000, acpt_ip2= 0000, acpt_ip1= 1000 and acpt_ip0=0101. Initially all accept counter (acpt_cntx) are 00. In case of input0 (acpt_ip0=0101), there are two grants from output0 and output2 respectively. Accept counter (acpt_cnt=00) points to 0 so accept will be given to output0 and acpt_op0=0001 and accept counter points to the next output (acpt_cnto=01). Similarly depending on accept input and initial value of accept counter, accept output set like this; acpt_op3=0000, acpt_op2=0000, acpt_op1=1000, acpt_op0=0001 and set the accept counter like this; acpt_cnt3=00, acpt_cnt2=00, acpt_cnt1=00, acpt_cnt0=01 (Modulo N increment to grant output).Same process is repeated after next clock (at 200.0 ns.), with updated accept counter.

In case of 8x8 RRM scheduler block, all the signals are 8 bits i.e. (7 downto 0) instead of 4 bits (3 downto 0) in 4 x 4 RRM scheduler block and hence complexity of each sub block increases. There is no change in clkipdorrm sub block.

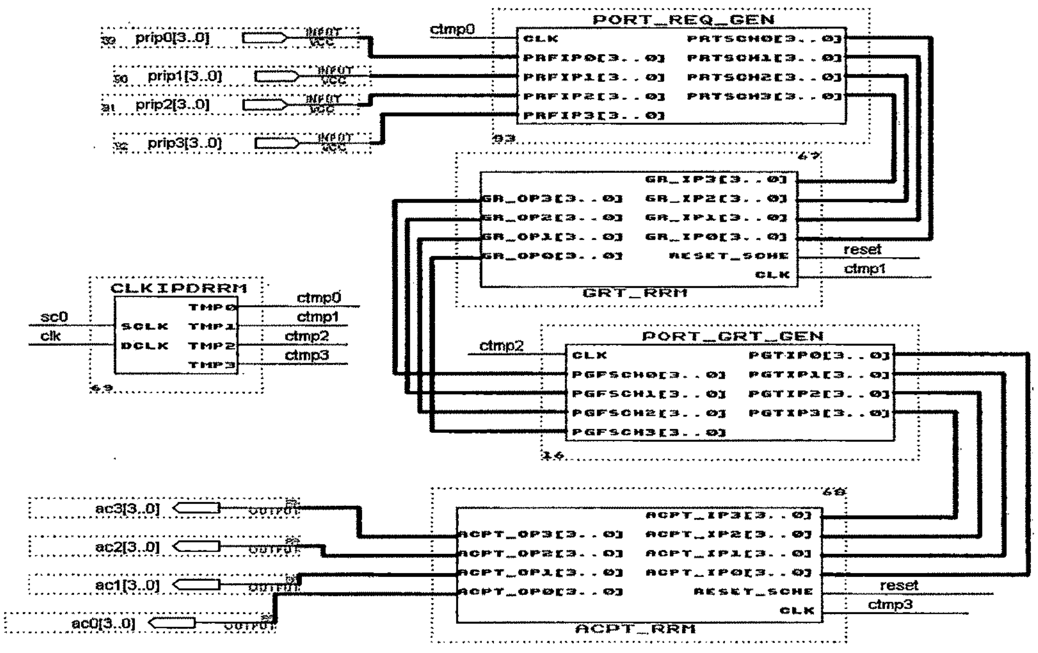


Figure 5.114 4x4 RRM scheduler block

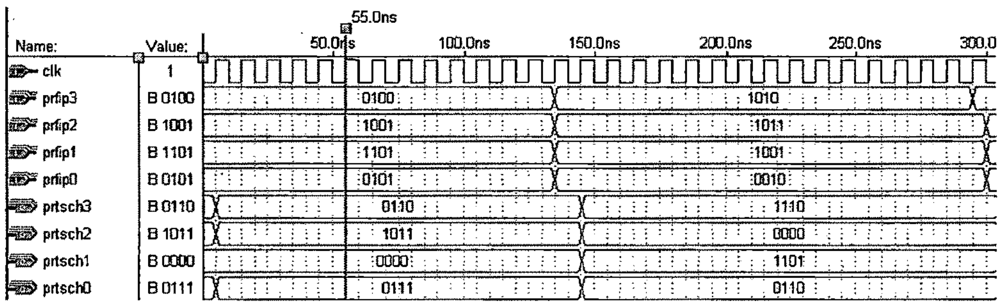


Figure 5.115 Simulation waveform of Port_Req_Gen

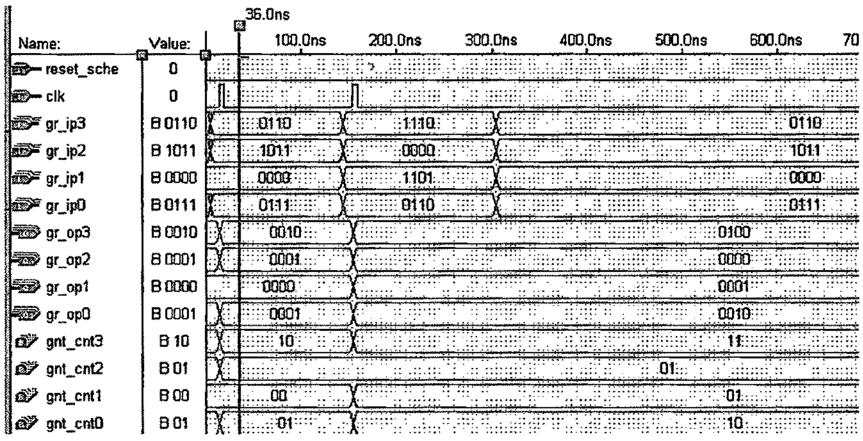


Figure 5.116 Simulation waveform of GRT_RRM

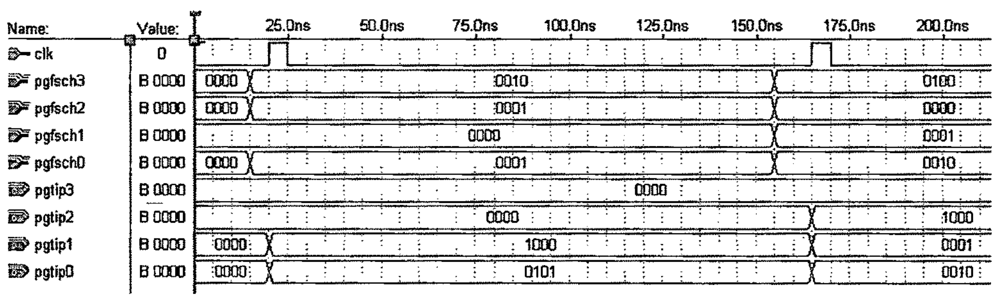


Figure 5.117 Simulation waveform of Port_GRT_Gen

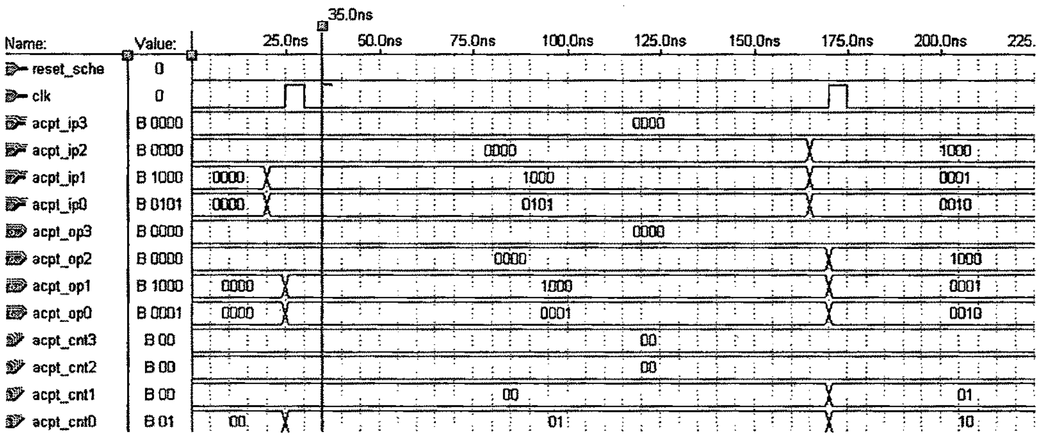


Figure 5.118 Simulation waveform of ACPT_RRM

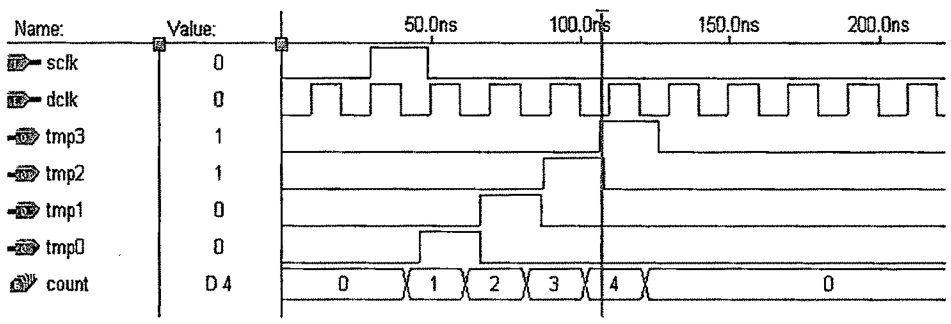


Figure 5.119 Simulation waveform of CLKIPDRM

5.6 VLSI IMPLEMENTATION OF iSLIP

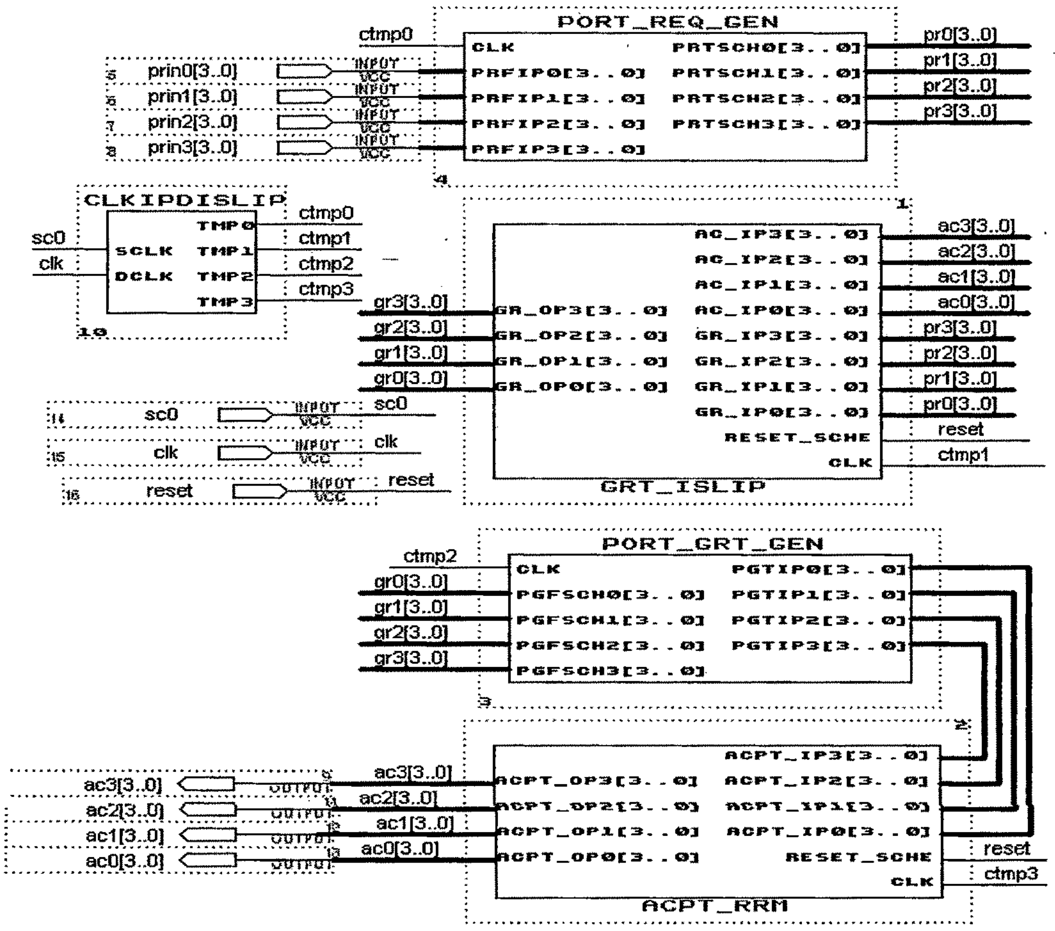


Figure 5.120 Internal detail of iSlip Scheduler

Figure 5.120 shows the detail of iSLIP Scheduler. Port_req_gen, Port _grt_gen and acpt_rrm blocks are same as RRM. Since iSLIP updates the grant pointer after confirmation of accept signal, we make change a small change in the grant_islip subblock. Grant phase now accepts final accept signals (acx[3..0]) to update the grant counter. To accommodate this change, clkipdislip block changes ctmp1 output (i.e clock of grt_islip block). Block clkipdislip produces second extra clock pulse at ctmp1 output to update the grant counter after update of accept signals. A grt_islip block produces, first grant output at the edge of first clock signal at ctmp1 and then waits for accept signals. After receiving accept signals, at the edge of second clock signal at ctmp1 it updates the grant counter, based on accept signals. So, if accept signal is not favorable, grant counter is not updated even though grant output is set. (gr2 [3..0]=0001, but accept is not given to that output so, grant counter is not updated to solve the synchronization problem.)

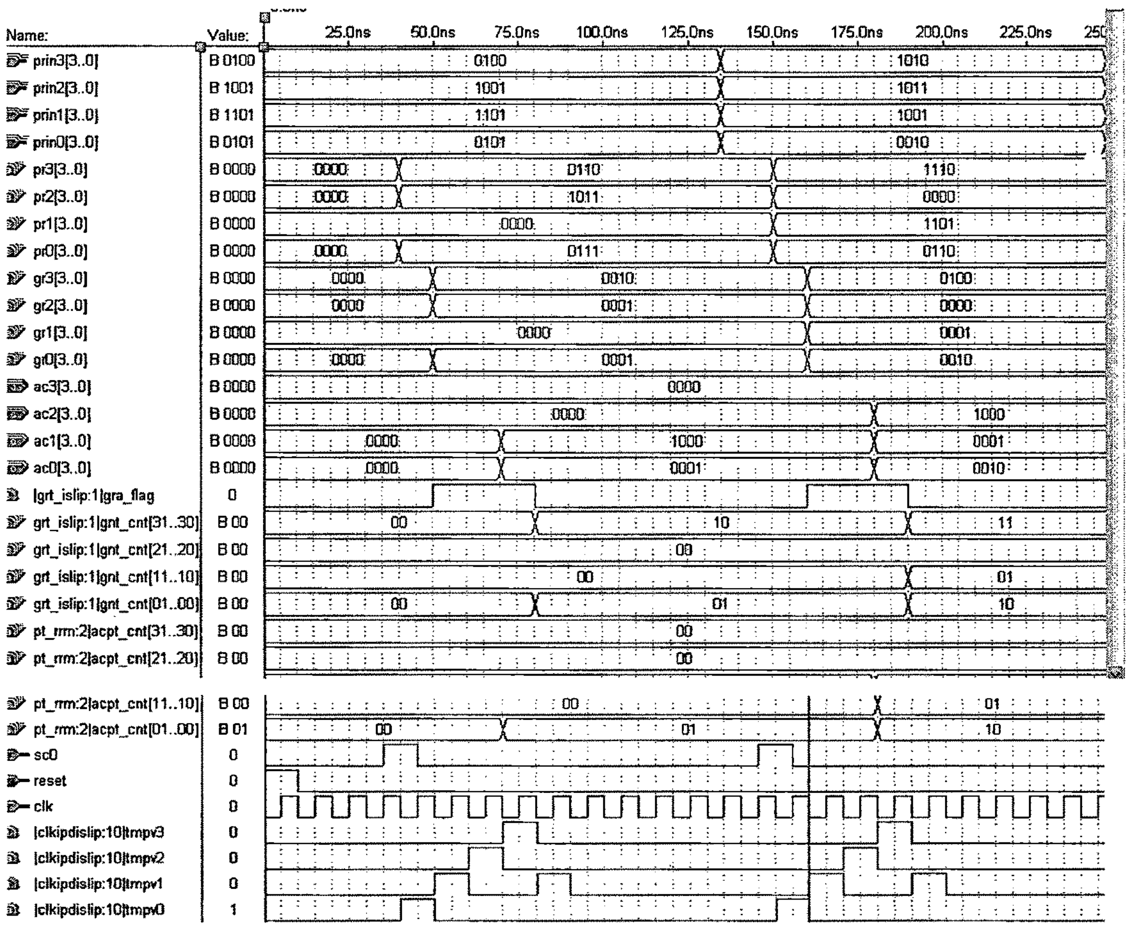


Figure 5.121 Simulation waveform of iSlip scheduler

5.7 VLSI IMPLEMENTATION OF RPA

In 4x4 RPA, 4 bit requests from all four inputs are converted to one 16 bit request signal *op_sched,* by req_all_ip sub block. Similarly 16 bit grant signal from scheduler_RPA is chopped to 4 grant signals (each of four bit) for each input. In RPA basic arbiter cell is made by using simple combinational logic as shown in figure 5.5. Such basic modular cells are arranged as shown in figure 5.7 by using component instantiation statements in VHDL. Due to modularity in structure, VLSI implementation of RPA is easy. As shown in figure 5.123 at 700ns c_bar_p=1111000 and c_bar_q=0011110 so priority starts from cell 3, 1. Our request input arb_req =fffff, so all the cells have requests. Now selected cells as per algorithm are (3,1),(4,2),(1,3),(2,4).Arrange same in the descending order (4,2),(3,1),(2,4),(1,3). So grant output is 2184H. (i.e. 0010, 0001, 1000, 0100 particular output bit is set 1).

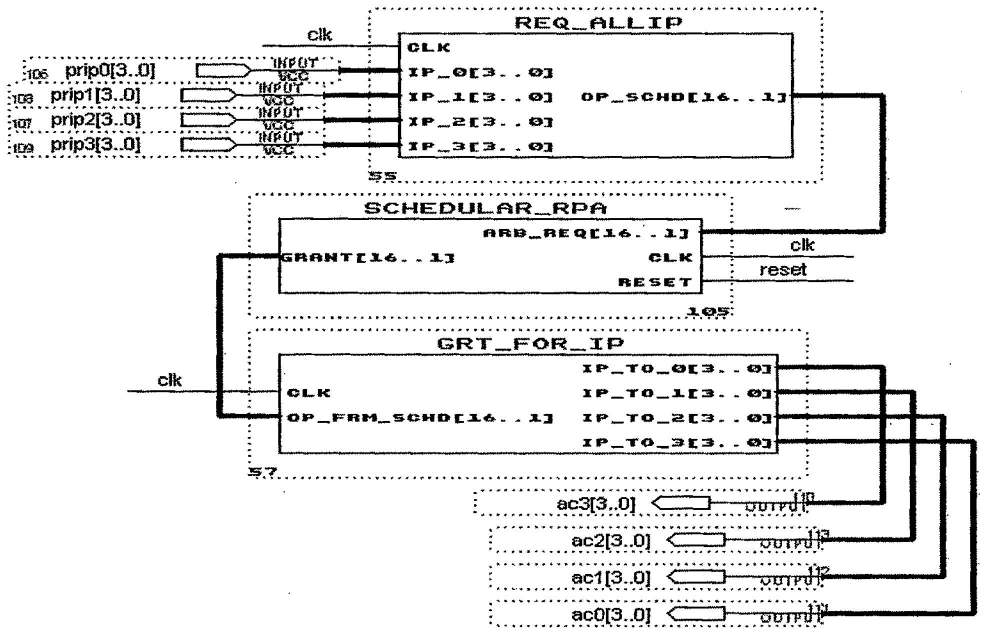


Figure 5.122 Internal detail of RPA Scheduler

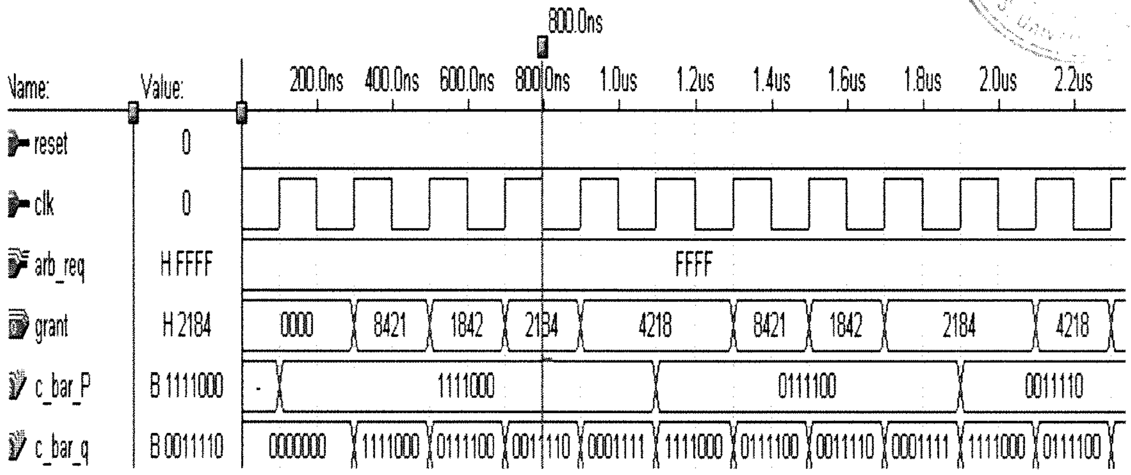


Figure 5.123 Simulation waveform of RPA_Schedular

5.8 VLSI IMPLEMENTATION OF DPA

Req_all_ip and grt_for_ip sub blocks in DPA are same as RPA. In DPA basic arbiter cell is made by using simple combinational logic as shown in figure 5.11. Such basic modular cells are arranged as shown in figure 5.10 by using component instantiation statements in VHDL. Due to modularity in structure VLSI implementation of DPA is easy.

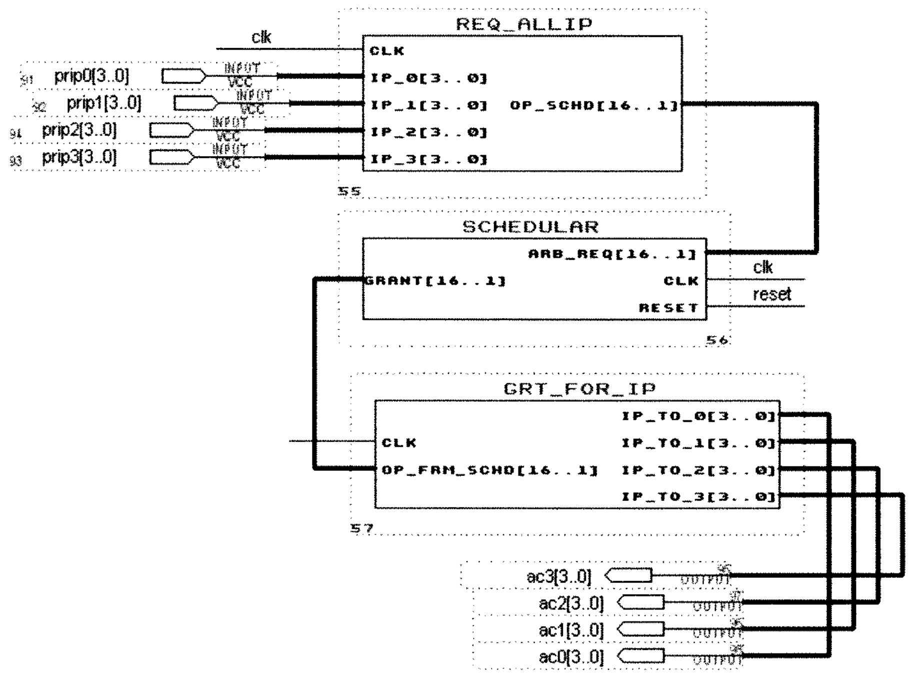


Figure 5.124 Internal detail of DPA Scheduler

As shown in figure 5.125 at 2us c_bar_p=0111100, so priority starts from cell 2,1. Our request input arb_req =fffff, so all the cells have requests. Now selected cells as per algorithm are 2,1, 1,2 , 4,3 , 3,4. We arrange same in descending input order i.e. (4,3),(3,4),(2,1),(1,2). So grant output is 4812H (i.e. 0100, 1000, 0001, 0010) particular output bit is set 1.

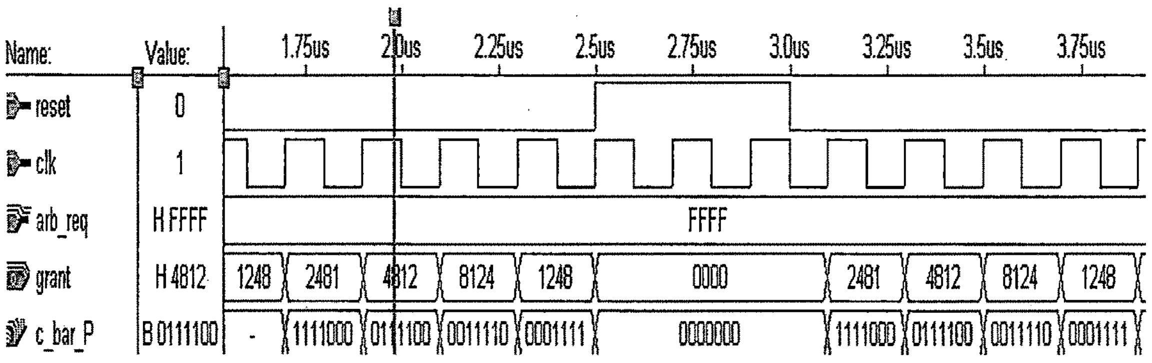


Figure 5.125 Simulation waveform of Scheduler for DPA

5.9 IMPLEMENTATION RESULTS

We implement the design in VHDL using ALTERA's MAX+PLUS II /Quartus tool. The VLSI area analysis of all the above scheduling algorithms is tabulated below.

Table 5.2 RRM VLSI area requirement for 4x4 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_RRM_4x4	EP20k1500 EBC652-1	5722 / 51,840 (11 %)	79 / 488 (16 %)	33,920 / 442,368 (8 %)	37. 92

Table 5.3 RRM VLSI area requirement for 8x8 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_RRM_8x8	EP20k1500EB C652-1	26,380 / 51,840 (51 %)	178 / 488 (36 %)	133,376/442,368 (30%)	21.72

Table 5.4 iSlip VLSI area requirement for 4x4 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_iSLIP_4x4	EP20k1500 EBC652-1	5707 / 51,840 (11 %)	87/ 488 (18 %)	33,920 / 442,368 (8 %)	39.70

Table 5.5 iSlip VLSI area requirement for 8x8 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_iSLIP_8x8	EP20k1500EB C652-1	26,067 / 51,840 (50 %)	178 / 488 (36 %)	133,376/442,368 (30%)	21.08

Table 5.6 RPA VLSI area requirement for 4x4 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_RPA_4x4	EP20k1500EB C652-1	5,810 / 51,840 (11 %)	87 / 488 (18 %)	33,920 / 442,368 (8 %)	10.51

Table 5.7 RPA VLSI area requirement for 8x8 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_RPA_8x8	EP20k1500E BC652-1	22,855 / 51,840 (44 %)	178 / 488 (36 %)	133,376 / 442,368 (30 %)	6.78

Table 5.8 DPA VLSI area requirement for 4x4 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_DPA_4x4	EP20k1500E BC652-1	5,656 / 51,840 (11 %)	87 / 488 (18 %)	33,920 / 442,368 (8 %)	13.59

Table 5.9 DPA VLSI area requirement for 8x8 ATM switch

Project	Device	Total Logic Elements	Total Pins	Total Memory Bits	Maximum Clock Frequency (MHz)
ATM_DPA_8x8	EP20k1500E BC652-1	22,778 / 51,840 (44 %)	178 / 488 (36 %)	133,376 / 442,368 (30 %)	6.66

From the VLSI area analysis, iSlip and RRM area requirements are more or less same but from the MATLAB simulation results, iSlip performance is far better than RRM. VLSI implementation of RPA and DPA is feasible due to their modular structure. VLSI area requirement of DPA is less than RPA but its performance is better than RPA as discussed in simulation.

5.10 SUMMARY

In this chapter, we survey basic scheduling algorithms and discuss MATLAB simulation of 4x4, 8x8, 16x16, and 32x32 crossbar switches. We generate 4x4, 8x8, 16x16, and 32x32 data for traffic pattern A, B, C, D and used this data as stimuli to PIM, RRM, iSLIP, RPA, and DPA algorithms. Throughput (efficiency) of DPA and RPA is higher than other algorithms, at the same time average latency of DPA and RPA is the lowest. Delay variance of DPA is less in traffic pattern A, C, and D, while delay variance of iSLIP is less in traffic pattern B.

We implement the 4x4 and 8x8 crossbar switches along with different scheduling algorithms in VHDL using ALTERA's MAX+PLUS II /Quartus tool. We test the functionality of each individual block, as well as the 4x4 and 8x8 overall switch design using VHDL simulations and observe a correct functional and timing performance. DPA has lowest area requirement and highest throughput, but iSLIP and RRM have higher maximum clock frequency. The simulations are run on a PC platform with a 3.2 GHz Pentium IV processor and a Windows XP operating system.