Designing of Efficient iSLIP Arbiter using iSLIP Scheduling Algorithm for NoC

Deepali Mahobiya

Department of Electronics & Communication Engineering ,RKDF College of Science & Technology,Bhopal M.P India

Abstract- Arbiters are a fundamental component in systems containing shared resources, and a centralized arbiter is a tightly integrated design for its input requests. In this study, we propose a new centralized arbiter, which may be used in arbitration of a crossbar switch in NoC routers. As fabrication technology continues to improve, smaller feature sizes allow increasingly more integration of system components onto a single die. Communication between these components can become the limiting factor for performance unless embodying the correct scheduling algorithm. In this paper, we design *i*SLIP arbiter using *i*SLIP scheduling algorithm with mesh router for NoC. An iterative, round-robin algorithm, *i*SLIP can achieve 100% throughput for uniform traffic, yet is simple to implement in hardware. Prototype and commercial implementations of *i*SLIP exist in systems with aggregate bandwidths ranging from 50 to 500 Gb/s. When the traffic is nonuniform, *i*SLIP quickly adapts to a fair scheduling policy that is guaranteed never to starve an input queue. This new arbiter is fair for any input combinations and faster & it is designed using ModelSimSE 6.3f for simulation of code and xilinx for synthesis.

Index Terms- *i*SLIP, iterative, scheduling algorithm, mesh router, arbiter.

I. INTRODUCTION

Following the rapid technological evolution, the complexity becomes one of the most constraining aspects in the design of embedded systems. Cost and timing issues come along to add to the difficulties in realization of network-on-chip, NoC, applications, where many IPs (Intellectual Property) such as processor cores, memories, DSP processors and peripheral devices are placed together, on a single die. These modules communicate, most often, by means of a shared resource, the on-chip network. The increasing complexity of the individual devices, the increasing demand for higher bandwidth on the network lines and an operating frequency hitting new limits with almost every new design, place the communication and/or computation resources arbitration being the performance bottleneck of the NoC system. The arbitration is desired to be completed within one clock cycle to avoid large latencies between the cores on the chip (*e.g.*, between a processing element (PE) and a memory block). To achieve the arbitration in one clock cycle, the overall delay introduced by the arbitration should be low so that it will not impact the overall system clock frequency, which introduces new challenges for the design of the arbiters.

In recent years, there has also been a significant amount of research work on scheduling algorithms for crossbar switches that use virtual output queuing.Scheduling algorithms have been developed by a number of researchers [5][6][7][8], for research prototypes [2][4], and commercial products [3]. What makes a good crossbar scheduling algorithm for the backplane of a router?

We desire algorithms with the following properties:

• *High Throughput* — An algorithm that keeps the backlog low in the VOQs(virtual output queueing). Ideally, the algorithm will sustain an offered load up to 100% on each input and output.

• *Starvation Free* — The algorithm should not allow any VOQ to be unserved indefinitely.

• *Fast* — To achieve the highest bandwidth switch, it is important that the scheduling algorithm does not become the performance bottleneck. The algorithm should therefore select a crossbar configuration as quickly as possible.

• *Simple to implement* — If the algorithm is to be fast in practice, it must be implemented in special-purpose hardware; preferably within a single chip.

Satisfying the above critereons here we present *i*SLIP arbiter using *i*SLIP scheduling algorithm with on-chip mesh router. The rest of the paper is organized as follows, in Section II, we briefly describe a generic RRA(Round Robin Arbiter) and identify issues limiting its scalability. In Section III, we introduce the proposed *i*SLIP Arbiter and its interconnections . Section IV describes the *i*SLIP scheduling algorithm. Section V concludes the paper.

II. PROBLEM DEFINITION

A. A Generic Round Robin Arbiter (RRA)

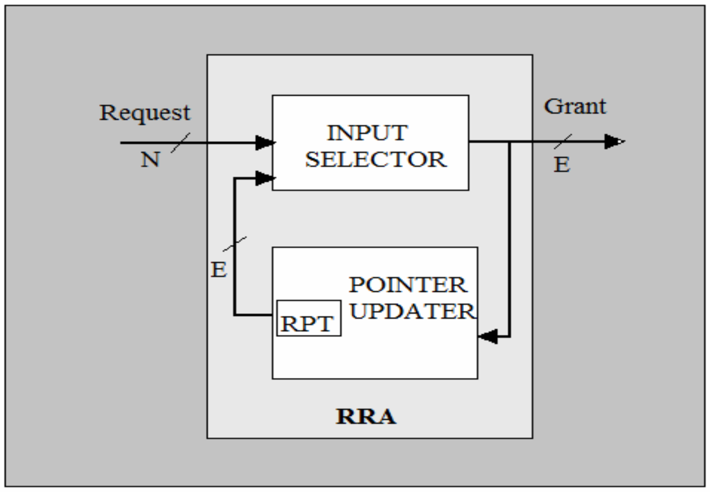


Figure 1: Generic Round Robin Arbiter architecture.

Given a set of requests from a set of inputs for a resource,the task of an RRA is to grant requests one input at a time in a fairly manner. A generic RRA consists of two main subblocks as shown in Figure 1: (1) Input Selector, and (2) Pointer Updater. The *Request* input port represents the *N* inputs with up to a total of *N* requests, and *Grant* output port represents the granted input with *E* = log2(*N*) bits. To achieve fairness among inputs, the RRA keeps an *E*-bit pointer (RPT) to the next possible input that would be granted in the next requestgrant cycle. The operation of the RRA is as follows. At the beginning of every clock cycle, each input with a request sets the corresponding bit in the Request to high. Input Selector decides the input that would be granted next, based on the Request and the current value of RPT. If there is a request from the input pointed by RPT (*i.e.*, Request [*RPT*] = *high*), this input will be granted by setting the Grant output value to the index of this input, *g*. If there is no request from the input pointed by RPT, but there are other requests, the Input Selector grants the first input with a request following the pointer in a circular manner (*i.e.*, if there is any request from inputs *i* > *RPT*, the smallest indexed input, *g* > *RPT* with a request will be granted. If there are no requests from inputs *i* > *RPT*, but there are requests from inputs, *i* < *RPT*, the smallest indexed input, *g* 6= *RPT* with a request, will be granted.). At the end of the clock cycle, the Pointer Updater sets the pointer value to the input next to the granted input in a circular manner (*i.e., RPT* = (*g* + 1) mod *N*). If there is no request from any inputs, the *RPT* will not change. An optional no request (*NoReq*) output port can be added to the generic RRA, which will be high when no input has a request.

B. Limitations on the Scalability of RRA

The most time-consuming operation of the generic RRA is the granting of the requests by the Input selector, which also dominates the critical path delay. The complexity of the Input Selector are due to two main issues: (1) *The issue of changing priority*:The priority of the inputs changes as inputs are granted and the RPT value changes. This requires the Input Selector circuit to consider all possible priority settings. (2) *The issue of circular priority order*: The priority order is circular, which makes the priority processing even harder. This leads to two separate conditions for the grant decisions: Grant decisions for requests at or below the Request [*RPT*] and grant decisions for requests above Request [*RPT*]. Any grant produced by the former decision has a higher priority over any grant produced by the latter decision. This two parted decision deepens the critical path. These two issues are even more pronounced as the RRA size gets larger (*i.e.*, an RRA with more inputs). Also RRA has its throughput near about 63% .

III. THE ISLIP ARBITER & ITS INTERCONNECTIONS

In this study, we propose a new centralized arbiter, which may be used in arbitration of a crossbar switch in NoC routers.The *i*SLIP arbiter is similar to Round-Robin arbiter, but with one change that here we keep a track of the requesting node and the grant given to the nodes previously, so that no node needs to wait for long time to get a grant. Maximum waiting period would be 1 clock cycle. *i*SLIP arbiter would keep a variable to check if this particular node has been assigned the grant signal. Each node gets the grant signal for one clock cycle, via which the node can transfer data in wireless or wired networks to other nodes.Figure. 2 shows the block diagram of an *n* × *n* switch and a crossbar switch fabric implemented with many transmission gates as switches between input ports and output ports. Each input port contains *n* virtual output queues (VOQs) to avoid head-of-line blocking. The task of the arbiter is to decide a set of contention-free connection between input and output ports by turningON/OFF those transmission gates. In general, the goal of a switch arbiter in a packet switch is to provide control signals to the crossbar switch fabric as shown in Figure. 2.

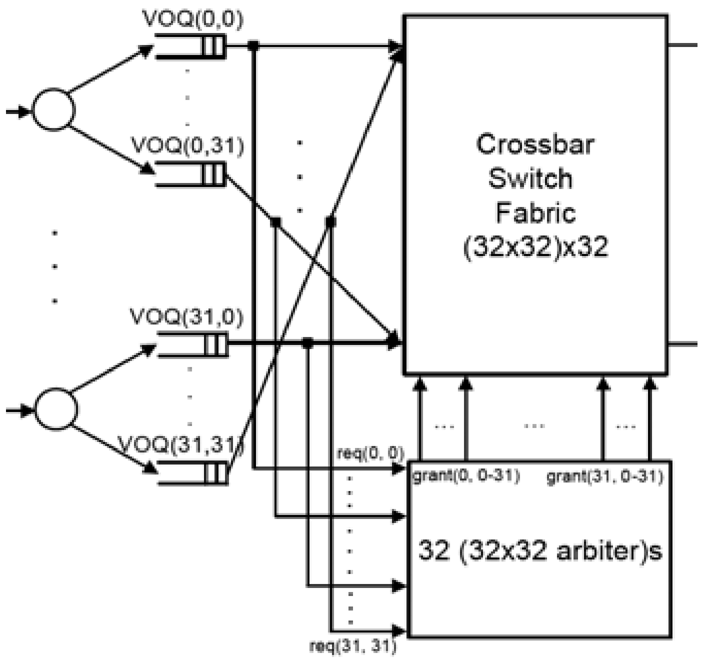


Figure 2: An *n* × *n* switch: block diagram, crossbar switch fabric and its switch arbiters.

Algorithm to configure the crossbar :

- Centralized scheduler serving the N2 VOQs

– Content of the VOQs examined at beginning of each cell time.
• Based on *i*SLIP arbitration
– arbiters schedule cells at each output and at each input.
– Input makes request to every output for which it has a queued cell.
– Output grants request that appears next in schedule.
– Input accepts grant that appears next in schedule.
– When arbitration is completed, matching between input and output has been found.

Figure. 3.[1] shows an example baseline implementation of the *i*SLIP algorithm. This implementation requires an NxN bit memory to contain the states of the input queues (N inputs x N output queues per input), N Grant Arbiters, N Accept Arbiters, and decision register to send control information to the datapath. The Grant and Accept Arbiters each consist of programmable priority encoders and a state pointer to record which input should have the highest priority on the next arbitration cycle. The N-bit feedback signal from the Decision registers to the Grant arbiters is an enable vector, which enables arbitration only for unmatched ports on each successive iterations.

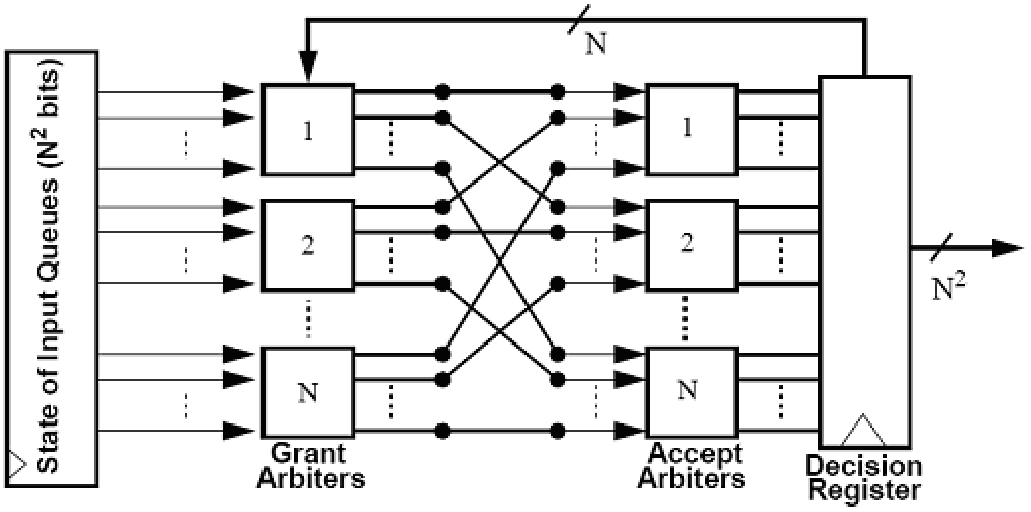


Figure 3:[1] 2N arbiters (N input and N output) interconnected to construct an *i*SLIP scheduler for an NxN switch Decision register configures the crossbar

The arbiter is responsible for configuring the input_blocks and the crossbar muxes to maximize utilization of the crossbar for each transmit cycle. Each data block indicates which destinations it has pending packets for, and the scheduler arbitrates to match inputs to destinations using the *i*SLIP scheduling algorithm.The crossbar interconnect uses a simple mux-based crossbar to enable a completely synthesized design. Once the scheduler configures the mux selects and data block destinations, the input_blocks simply pump the data through the interconnect to the output_blocks over multiple cycles. Once the output block receives the full packet, it transmits the packet to the connected output device. *i*SLIP is good for high speed as its input buffers are separated. For each input and output there is a Separate scheduler & each works independently.

IV. THE ISLIP SCHEDULING ALGORITHM

A. iSLIP ALGORITHM

The *i*SLIP algorithm is designed to meet our goals. *i*SLIP is an iterative algorithm — during each time slot, multiple iterations are performed to select a crossbar configuration, matching inputs to outputs. The *i*SLIP algorithm uses rotating priority (“round-robin”) arbitration to schedule each active input and output in turn. The main characteristic of *i*SLIP is its simplicity; it is readily implemented in hardware and can operate at high speed. A prototype version of *i*SLIP is currently being implemented that makes a new scheduling decision every 40ns[10]. *i*SLIP attempts to quickly converge on a conflict-free match in multiple iterations, where each iteration consists of three steps. All inputs and outputs are initially unmatched and only those inputs and outputs not matched at the end of one iteration are eligible for matching in the next. The algorithm for *i*SLIP, taken from [1]. The steps of each iteration 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.

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. The pointer to the highest priority element at the corresponding output is incremented (modulo N) to one location beyond the granted input. The pointers are only updated *if and only if* the grant is accepted after the first iteration. By considering only unmatched inputs and outputs, each iteration matches inputs and outputs that were not matched during earlier iterations.

The priority selector circuit is shown in Figure.4 . which decides the highest and lowest priority element.

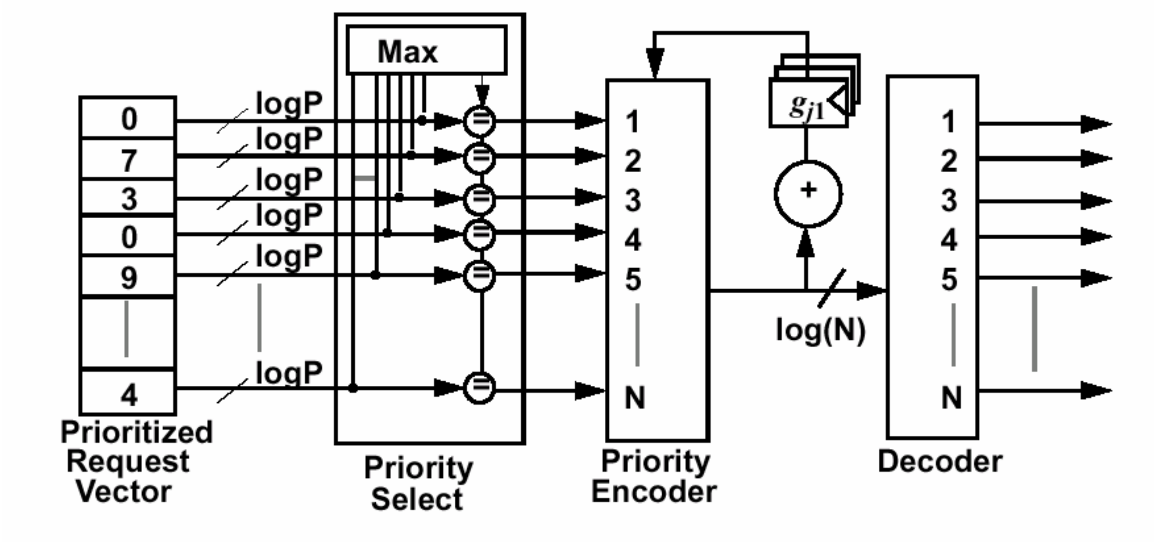


Figure 4: Priority selector circuit

B. PERFORMANCE

Despite its simplicity, the performance of *i*SLIP is surprisingly good. A detailed study of its performance and implementation can be found in [9], and more briefly in [10]. In brief, its main properties are:

Property 1. *High Throughput* — For uniform, and uncorrelated arrivals, the algorithm enables 100% of the switch capacity to be used.

Property 2. *Starvation Free* — No connection is starved. Because pointers are not updated after the first iteration, an output will continue to grant to the highest priority requesting input until it is successful. Furthermore *i*SLIP is in some sense fair: with one iteration and under heavy load, all queues with a common output have identical throughput.

Property 3. *Fast* — The algorithm is guaranteed to complete in at most *N* iterations. However, in practice the algorithm almost always completes in fewer than iterations. i.e. for a switch with 16 ports, four iterations will suffice.

Property 4. *Simple to implement* — An *i*SLIP scheduler consists of 2*N* programmable priority encoders. A scheduler for a 16-port switch is readily implemented on a single chip.

The performance of *i*SLIP can be seen in Figure.5. It shows that throughput of RRM is increaced from 63% to 100% in *i*SLIP.

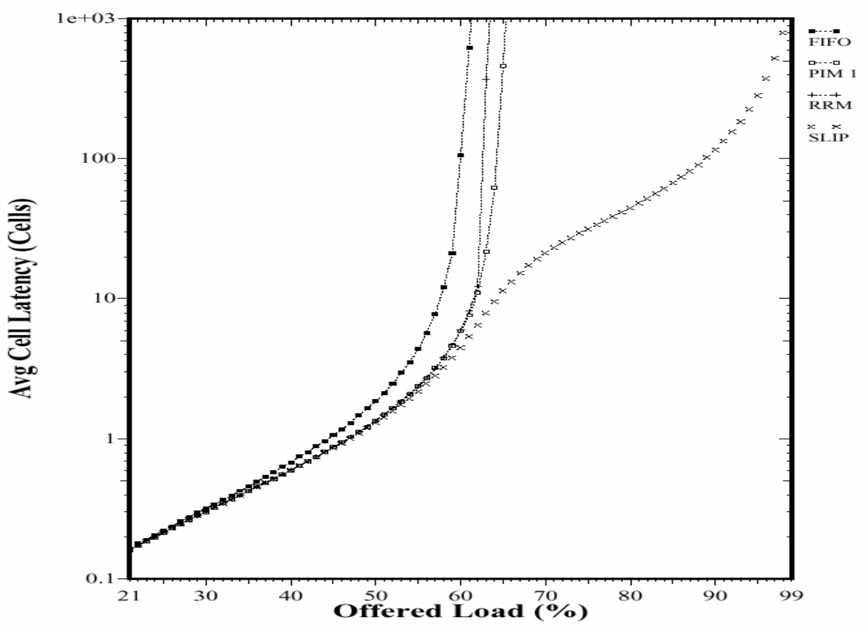


Figure 5: Performance graph

V. CONCLUSION

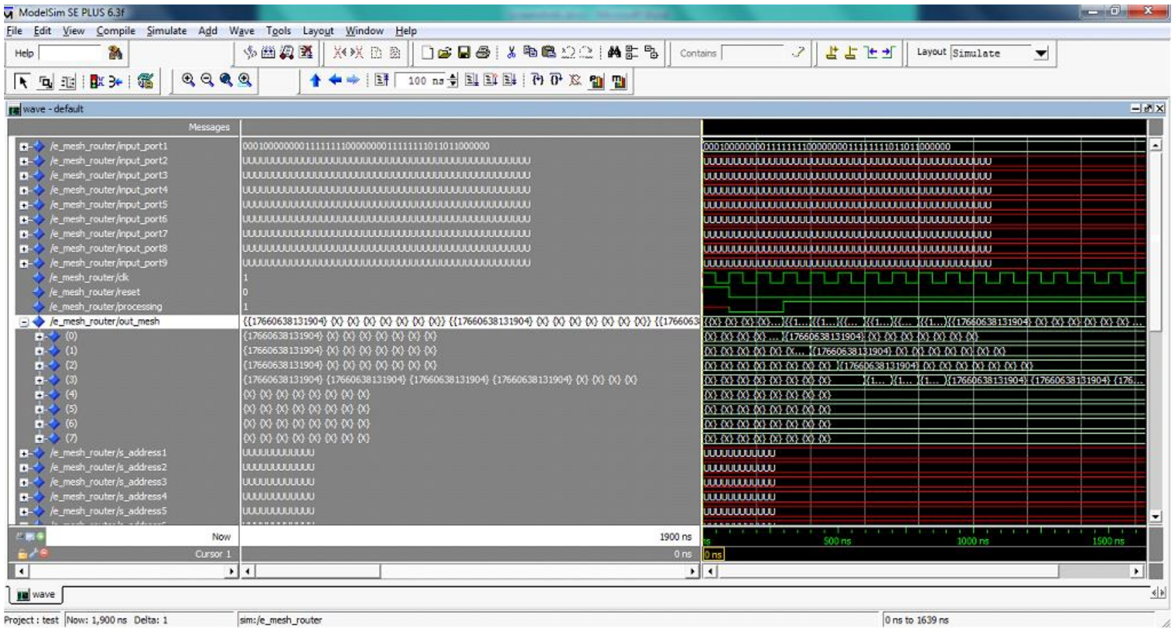
The recent advances in Networks-on-Chip (NoC) introduced new challenges for switches/routers for NoC. *i*SLIP arbiters, crucial building blocks for switches/routers, are required to complete an arbitration within one short clock cycle to provide low-latency links between processing elements and memory blocks. In this paper, we proposed *iSLIP Arbiter & iSLIP scheduling algorithm for on-chip mesh router* , a high-speed, high throughput, starvation free and easy to implement solution for arbitration in NoC. The arbiter trap the source and destination address from the output 0f buffer and generate the control signal so that input data from source side sending to the output port. The following Appendix [A] provide the simulation results of the arbiter. Where we can see that the input packet is being routed in the network from node 1 to node 3. The results indicate the same.

ACKNOWLEDGMENT

Along the way, the development of *i*SLIP first of all, I would like to thank my supervisor Professor Tejasvini Choudri for the encouragement and guidance which she provided me throughout this project. I had a great opportunity of learning so many things from her in the meetings and brainstorming sessions. It is worthy to mention that it is an honor for me to accomplish master thesis. I would like to thank Ashwin sir for providing a ModelSim simulator that was used in simulation. I thank him for his time and effort which he spent in debugging the code for hours. Without his help, the simulation results were not possible in such a small span of time.I would like to thank all my teachers for delivering great and invaluable knowledge. They are true educators and possess great knowledge.

APPENDIX A: SIMULATION RESULT

Here we can see that the input packet is being routed in the network from node 1 to node 3.



```
/e_mesh_router/input_port1     000100000000111111110000000011111111011011000000
/e_mesh_router/input_port2     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port3     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port4     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port5     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port6     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port7     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port8     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/input_port9     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/e_mesh_router/clk             1
/e_mesh_router/reset           0
/e_mesh_router/processing      1
/e_mesh_router/out_mesh        {{17660638131904} {X} {X} {X} {X} {X} {X} {X}} {{17660638131904} {X} {X} {X} {X} {X} {X} {X}} {{1766063
  (0)                          {17660638131904} {X} {X} {X} {X} {X} {X} {X}
  (1)                          {17660638131904} {X} {X} {X} {X} {X} {X} {X}
  (2)                          {17660638131904} {X} {X} {X} {X} {X} {X} {X}
  (3)                          {17660638131904} {17660638131904} {17660638131904} {17660638131904} {X} {X} {X} {X}
  (4)                          {X} {X} {X} {X} {X} {X} {X} {X}
  (5)                          {X} {X} {X} {X} {X} {X} {X} {X}
  (6)                          {X} {X} {X} {X} {X} {X} {X} {X}
  (7)                          {X} {X} {X} {X} {X} {X} {X} {X}
/e_mesh_router/s_address1      UUUUUUUUUUUU
/e_mesh_router/s_address2      UUUUUUUUUUUU
/e_mesh_router/s_address3      UUUUUUUUUUUU
/e_mesh_router/s_address4      UUUUUUUUUUUU
/e_mesh_router/s_address5      UUUUUUUUUUUU
Now                            1900 ns
Cursor 1                       0 ns
```

```
Project : test   Now: 1,900 ns  Delta: 1   sim:/e_mesh_router   0 ns to 1639 ns
```

REFERENCES

[1] McKeown, N. W. 1995 *Scheduling Algorithms for Input-Queued Cell Switches*.Doctoral Thesis. UMI Order Number: UMI Order No. GAX96-02658., University of California at Berkeley.

[2] C. Partridge; P. Carvey et al. "A Fifty-Gigabit per Second IP Router," Submitted to IEEE/ACM Transactions on Networking, 1996.

[3] T. Anderson et al. "High Speed Switch Scheduling for Local Area Networks," ACM Trans. on Computer Systems, Nov 1993, pp. 319-352.

[4] N. McKeown et al. "The Tiny Tera: A small high-bandwidth packet switch core," IEEE Micro, Jan-Feb 1997.

[5] LaMaire, R.O.; Serpanos, D.N.; "Two-dimensional round-robin schedulers for packet switches with multiple input queues," *IEEE/ACM Transactions on Networking*, Oct. 1994, vol.2, (no.5):471-82.

[6] Lund, C.; Phillips, S.; Reingold, N.; "Fair prioritized scheduling in an input-buffered switch," *Proceedings of the IFIP-IEEE Conf. on Broadband Communications '96*, Montreal, April 1996. p. 358-69.

[7] Hui, J.; Arthurs, E. "A broadband packet switch for integrated transport," *IEEE J. Selected Areas Communications, 5*, 8, Oct 1987, pp 1264-1273.

[8] Ali, M.; Nguyen, H. "A neural network implementation of an input access scheme in a high-speed packet switch," *Proc. of GLOBECOM 1989*, pp.1192-1196.

[9] N. McKeown, "Scheduling Cells in an input-queued switch," PhD Thesis, University of California at Berkeley, May 1995.

[10] N. McKeown, "*i*SLIP: A Scheduling Algorithm for Input-Queued Switches," Submitted to IEEE Transactions on Networking.

AUTHORS

First Author – Deepali Mahobiya, Degree: Bachelor of Engineering in Electronics & communication, Pursuing M-Tech from R.K.D.F College of science & Techonology, Bhopal M.P India, email address:- thakur.deepali1@gmail.com

Correspondence Author – Mrs.Deepali Thakur Mahobiya, thakur.deepali1@gmail.com, 07819828600.