TIME EFFICIENT ARBITER IN THE DESIGN OF SCHEDULER EMBODYING ISLIP ALGORITHM FOR ON-CHIP INTERCONNECION

Vilas N. Nitnaware
Department of Electronics Design Technology,
Shri Ramdeobaba K. N. Engg. College, Nagpur, India.
nitnawarevn@rknec.edu
Shyam S. Limaye
Principal, Jhulelal Institute of Technology,
Nagpur, India.
Shyam_limaye@hotmail.com

Abstract

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 careful attention is given to designing high performance interconnects. Amongst various components of the interconnect, a high-performance arbiter in a scheduler decides the speed of scheduling. An intelligent centralized scheduler is needed to configure the crossbar fairly and with high utilization.

The main contribution of this paper is the design and optimization of fast round- robin arbiters and the design of a On-Chip Scheduler embodying I-SLIP algorithm. An iterative, round-robin algorithm, iSLIP can achieve 100% throughput for uniform traffic, yet is simple to implement in hardware. Iterative and noniterative versions of the algorithms are presented, along with modified versions for prioritized traffic. Scheduler is expressed here in verilog RTL and simulation results are presented to indicate the performance of iSLIP under benign and bursty traffic conditions.

Prototype and commercial implementations of iSLIP exist in systems with aggregate bandwidths ranging from 50 to 500 Gb/s. When the traffic is nonuniform, iSLIP quickly adapts to a fair scheduling policy that is guaranteed never to starve an input queue. We describe the implementation complexity of iSLIP algorithm in a round robin scheduler which configures 8x8 crossbar.

*Further, we have synthesized the accept and grant arbiters and optimized its area and timing using TSMC's library [tcb015ghdbc] with TSMC8k_Conservative wire load model. The request is processed pretty fast and reaches at grant output of the arbiter in **0.59 ns**. The total cell area of the proposed arbiter design is **445**.Further the scheduler is synthesized to obtain its cell area **20393** while the longest path takes **0.52 ns time.** It becomes the most optimized scheduler in an On-Chip Interconnect. The designs were optimized under the same operating conditions with similar area and timing constraints using TSMC's library [tcb015ghdbc].*

***Keywords**: Switch, Virtual queues, Thermometer encoding, PPE, state pointer, input block.*

1. The Scheduling Algorithm:

The algorithm for i-SLIP, taken from [**1**], follows:

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

Step 2: *Grant.* In an unmatched 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 g_i to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if the grant is accepted in Step 3 of the first iteration.

Step 3: *Accept.* If an unmatched 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 a_i to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output only if this input was matched in the first iteration.

Changes to i-SLIP

Due to the flow control mechanisms, i-SLIP must be modified so it doesn't connect an input port to an output that does not have enough credit remaining to accept a data transfer. To handle this case, outputs with 0 credit will not participate in the arbitration iteration. If credit arrives between iterations of the same scheduling cycle, the port may become active again and participate in the remaining arbitration iterations.

The *i*SLIP algorithm has the following properties [11]:

Property 1. For independent arrivals uniformly distributed over all outputs, *i*SLIP achieves 100% throughput with just a single iteration, with more iterations, the queueing delay is reduced.

Property 2. No connection is starved; because of the requirement that pointers are not updated after the first iteration, an output will continue to grant to the highest priority requesting input until it is successful.

Property 3. For *i*SLIP with one iteration, and under heavy load, queues with a common output all have the same throughput.

Property 4. The algorithm will converge in at most N iterations. Simulation suggests that on average, the algorithm converges in fewer than $\log_2 N$ iterations.

Property 5. The deterministic nature of *i*SLIP reduces the burstiness of traffic as it transits the switch.

2. Arbiters

The arbiters are an important piece of the scheduler design. The Grant Arbiters and Accept Arbiters are identically designed with the exception of the rules determining when the priority state may be updated. **Figure1** illustrates the arbiter design chosen for this implementation. The arbiter is based on a simple round-robin arbiter, with the exception that it also includes an update_enable signal to allow the i-SLIP algorithm to only update the priority under certain circumstances. This limited updating produces desynchronizing behavior between the Grant and Accept arbiters, producing improved traffic fairness and decreasing undesirable bursting characteristics.

As we see from the high-level block diagram in **Figure1**, the delay through the grant and accept arbiters directly affects the speed of the scheduling algorithm. To make the arbiters fast, we first observe that a round-robin arbiter is equivalent to a programmable priority encoder, plus some state to store the round-robin pointer. A programmable priority encoder (PPE) differs from a simple priority encoder in that an external input dictates which input has the highest priority. In what follows, we take a detailed look at the design of a high speed round robin arbiter.

It has some state (called round-robin pointer, *P_enc*, of width *lgN* bits), which points to the current highest priority input. In every arbitration cycle, it uses this pointer *P_enc* to choose one among the *N* incoming requests, through a programmable priority-encoder. This PPE takes in *N* 1-bit wide requests and an *lgN*-bit wide pointer (which we call *P_enc*) as inputs. It then chooses the first non-zero request value beyond (and including) *Req[P_enc],* resulting in an *N*-bit grant. Clearly, the core function of contention resolution is carried out by this combinational block. The pointer-update mechanism is generally simple and can be performed in parallel. To minimize overall delay, we focus on minimizing the path from *Req* to *Gnt*, which is a pure combinational path passing through the PPE. Hence, the problem of designing a fast round-robin arbiter is reduced to designing a fast PPE. It is to be noted that a fast PPE could be used in any arbiter, regardless of the pointer-update mechanism.

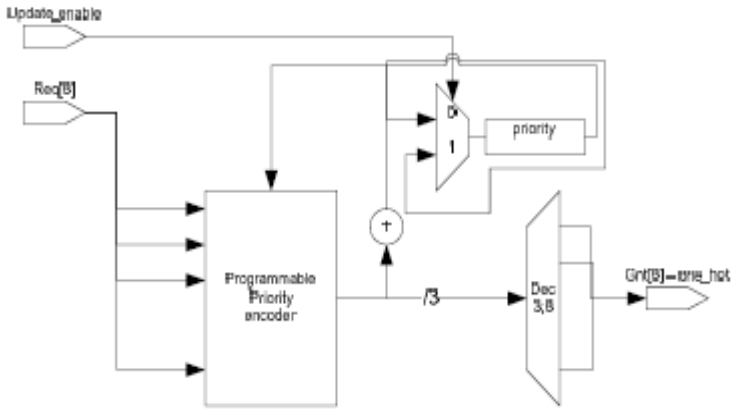


Figure 1

3. Programmable Priority Encoder

The most timing-critical component of the scheduler design is the Programmable Priority Encoder (PPE) utilized by each Arbiter. An 8-bit priority encoder takes 8 bits on inputs 0 thru

7, and outputs a number indicating the highest priority input that was active. A simple priority encoder always has a fixed priority (0 – highest, 7 – lowest), but a programmable priority encoder has an additional input indicating which input has the highest priority. The speed of the programmable priority encoder will determine how fast the arbitration logic can run, and is likely to be the critical path of the overall interconnect design[4].

The PPE chosen for this design is a hybrid design combining two simple PEs. The input to one of the simple PEs is masked by a thermometer encoding based on the programmed priority level [5]. If that PE provides any grant, it takes precedence over the non-masked simple PE. The non-masked simple PE does not take a priority, and determines the output when the masked PE does not find a 1-input between the programmed priority and input 7.

We first look at thermometer encoding, which we use in our design.
Thermometer encoding:

A thermometer encoding of a $\log_2(N)$-bit wide vector x, (i.e. $0 \leq value(x) < N$) is a N-bit wide vector y with the following transformation equation: $y[i] = 1$ *iff* $i < value(x)$, $0 \leq i < N$. The truth table for N=8 is shown in **Table1** as one example of the transformation.

Table1

X(2..0) -[Priority]	Y(7..0)- [Thermometer output]
000	00000000
001	00000001
010	00000011
011	00000111
100	00001111
101	00011111
110	00111111
111	01111111

We find that this transformation take no longer than decoding x to give a 1-hot vector *x_dec*. For example, **Figure 1(a)**shows the equa- tions for N=8. A fairly simple recursive algorithm can generate the equations for any N (that is a power of 2). Such an algorithm is sketched out in **Figure 1(a)**. As can be easily seen from the description of the algorithm, each *y[i]* is either an OR or AND of no more than "width=$\log_2(N)$" terms. This is the same complexity as decode logic.

Obtaining thermometer encoding for N = 8.

y7 = 0
y6 = x2.x1.x0
y5 = x2.x1
y4 = x2.(x1+x0)
y3 = x2
y2 = x2+x1.x0
y1 = x2+x1y0 = x2+x1+x0

Figure 1(a)

PPE works as follows : if there are no requests in the range *P_enc..N-1*, the output of the PPE is the same as the output of a simple priority encoder *smpl_pe*. If there is a request in the range *P_enc..N-1*, then the output of the PPE equals the output of *ppe_not_round*. This observation implies a simple 2:1 multi- plexer for each of the *N*-bits of the output vector *Gnt*, with *anyGnt_not_round* as the select signal. The multiplexer is utilizing the fact that when *anyGnt_not_round* is '0', *Gnt_not_round* will be a zero vector, and so can directly by ORed to give the final output. This reduces the loading on the select signal (*anyGnt_not_round*) by half. Design PPE_comb does not have any combinational feedback loop and can be fully optimized and tested by automated tools.

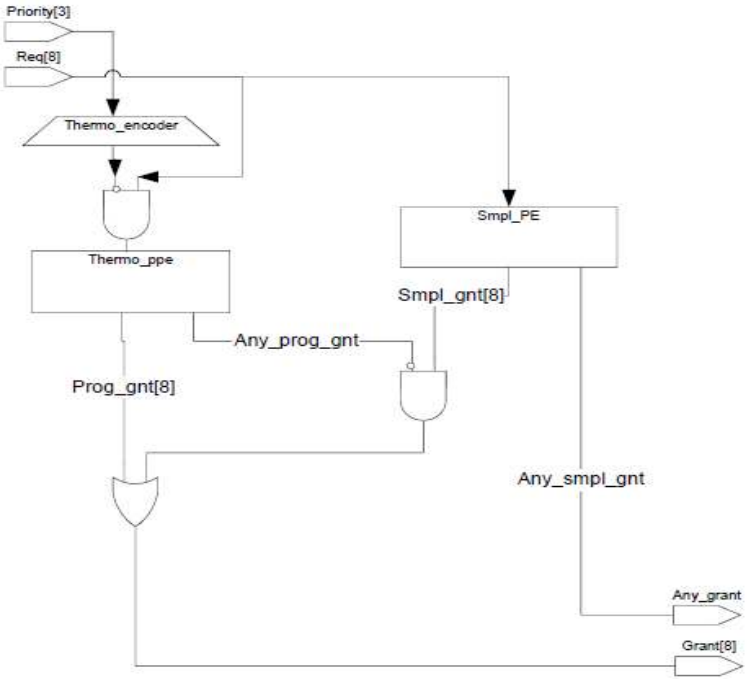


Figure 2

4. Scheduler

The Scheduler acts as the central switch arbiter. It analyzes the occupied Virtual Output Queues of each input_block and configures the input_blocks and interconnect muxes to connect inputs to outputs and allow serial data transfer across the switch. The scheduling algorithm attempts to achieve a large number of simultaneous connections, but also avoids conflicts of multiple inputs connecting to a single output or a single input connecting to multiple outputs. The scheduling algorithm chosen is a modified i-SLIP [1] scheduler.

The base i-SLIP specification comes from Chapter 3, Section 5 of [1]. The i-SLIP algorithm is derived from the SLIP algorithm[1], which is an improvement upon the Round-Robin Matching algorithm. Each of these algorithms assumes an N-input by N output cell switch with input queuing. To alleviate head-of-line blocking at the input queues, each input maintains a separate queue for each possible output destination. The goals for the scheduling

algorithm is to match input queues containing waiting packets with output queues to achieve the maximum throughput while maintaining stability and eliminating starvation.

The SLIP algorithm matches inputs to outputs in a single iteration; however, after this iteration, several possible input and output ports may remain unutilized. The i-SLIP algorithm uses multiple iterations to find paths to utilize as many input and output ports as possible (pseudo-maxsize matching) until it converges to finding no more possible matches. The single iteration SLIP algorithm is a specialization of i-SLIP and may be characterized as i-SLIP with only a single iteration, or 1-SLIP.

Figure 3 shows the i-SLIP scheduler implementation chosen for this design. The input to the scheduler is the occupancy vectors from each of the input_blocks with packets waiting to be scheduled. There are such 8 vectors (1 per crossbar input port), each with 8 bits (1 per destination per input). This 8x8 state is used as the request vector into the iterative arbitration logic.

The scheduler also contains 8 Grant Arbiters and 8 Accept Arbiters. 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 8-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.

Finally, after a number of iterations, the scheduler arrives at a final scheduling solution which it outputs to each of the input_blocks (indicating which destination the data block has been scheduled to transmit a packet for) and each interconnect mux (indicating which input_block it is receiving data from).

As an enhancement to the original i-SLIP algorithm proposed in [1], this scheduler also includes an 8-bit busy input from each of the switch outputs. These busy signals are asserted if that output does not have enough downstream credit to send another transfer. When the busy signal is asserted, that output port is disabled from the Grant Arbitration.

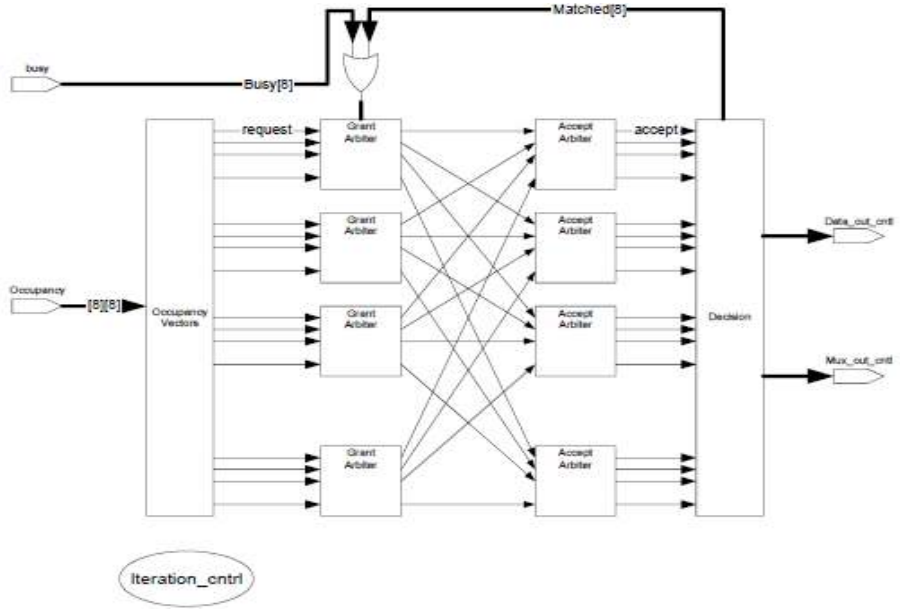


Figure 3

5. Constraints:

1. My implementation of the i-SLIP protocol uses a fixed number of input and output ports. I begin with an 8x8 design in synthesized static CMOS. Since the most complex portion of the design appears to be the programmable priority encoders in arbiters, most of the design phase required to focus on optimizing their implementation for speed and area.
2. The i-SLIP switch is programmable to support a variable number of iterations from 1 thru N. It shows that the maximum number of iterations required to converge is N.

The following are some of the microarchitectural optimizations used to improve timing.

Programmable Priority Encoder

The slowest timing path through the scheduler passes through the programmable priority encoder logic of the grant/accept arbiters. There are many different ways to implement a priority encoder that supports a programmable priority [5]. Some require rotating the request vector and then rotating the resulting priority vector, while others resemble a ripple-carry-adder design. However, I chose a design that utilizes a request masking mechanism and two simple non-programmable priority encoders for this implementation.

I chose to implement the simple priority encoder using a simple verilog casez statement. When Boolean optimizations were enabled in the DesignVision synthesis tool, it synthesized the casez into a tree-like structure, achieving very low delay through the priority encoder.

6. Testing

The Scheduler design underwent a large testing effort to verify functional correctness. The testing involved both unit and system-level testing using structured and random stimulus and assertions. The testing occurred in two phases: unit-level verification and system-level verification.

The scheduler unit verification was accomplished using a combination of directed input stimulus and assertions. The inputs to the scheduler are the occupancy vectors of each of the input blocks. For unit-level testing, short test cases that exercised the occupancy vectors were used to stimulate the scheduler inputs. The scheduling decisions were partially checked using assertion blocks to verify that the decisions conformed to all-0 or 1-hot encoding. For the unit-level scheduler test bench, no checking was done to verify that a correct schedule was determined given the current occupancy vectors. That checking was deferred to the system-level simulations.

7. Area and Timing Results

The start point of the design is req while the endpoint of the design is gnt, the data arrival time is 0.59 ns which is quite fast as compared to other PPEs proposed in [5]. The total cell area of the proposed arbiter design is 445 μm^2 using 150 nm, tcb015ghdbc.db[TSMC] Technology library. Further the scheduler is synthesized to obtain its cell area 20393 while the longest path takes 0.52 ns time. Design Compiler performed minimal area optimization as no area constraint was set[8].

Table 2: Times (in nanoseconds) for each arbiter design.

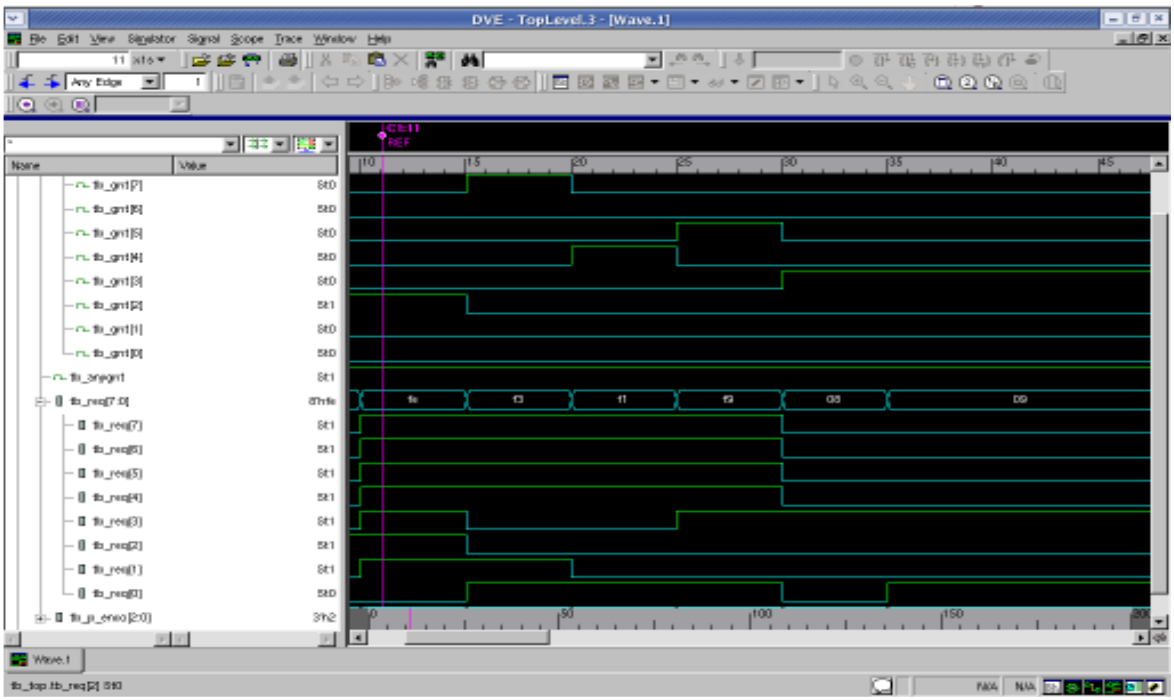
Design	RIPPLE	CLA	EXH	SHFT_ ENC	PROPOSED IN [5]	**PROPOSED IN THIS PAPER**
***N*=8**	3.72	2.53	1.11	2.13	1.01	**0.59**

8. Conclusion

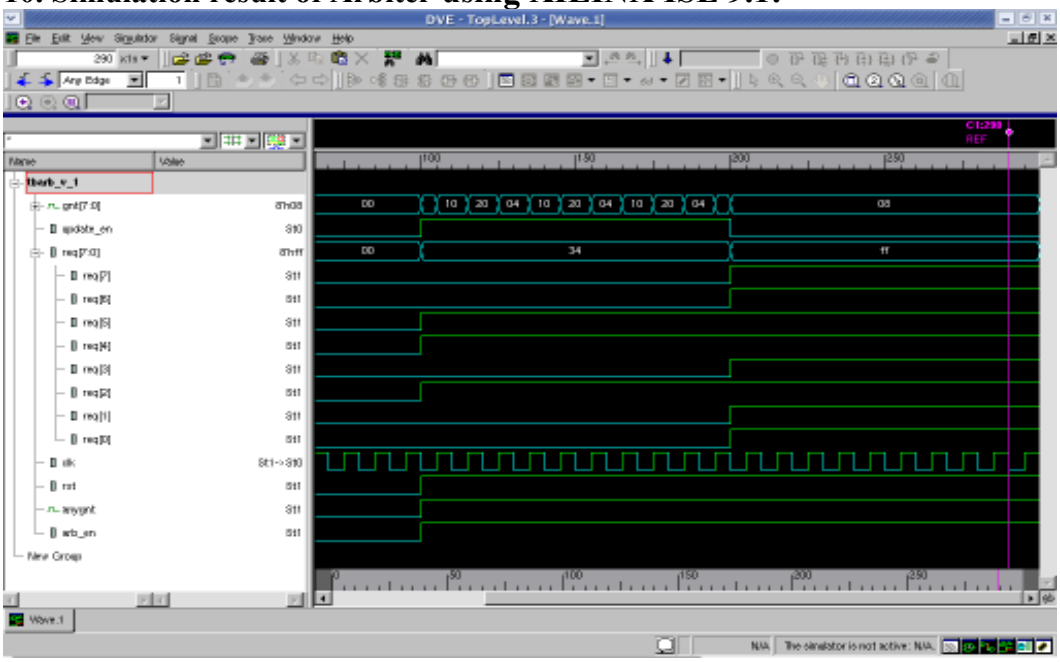
This project designed an 8x8 on chip scheduler module utilizing the iSLIP scheduling algorithm. The design was implemented in Verilog RTL and synthesized arbiters using Synopsys Design Vision and a 150nm synthesis library to achieve a minimum data arrival time of 0.59ns.The total cell area of the proposed arbiter design is 445 μm^2. Further the scheduler is synthesized to obtain its cell area 20393 while the longest path takes 0.52 ns time. It becomes the most efficient scheduler in an on chip interconnect compared to earlier implemented. Design Compiler performed minimal area optimization as no area constraint was set. Iterative and noniterative versions of the algorithms are presented, along with modified versions for prioritized traffic. The design was tested using unit-level and system level test benches utilizing random stimulus and strategically-placed assertions.

Design Compiler performed minimal area optimization as no area constraint was set[8]. By default the clock rises at 0ns and has a 50% duty cycle.

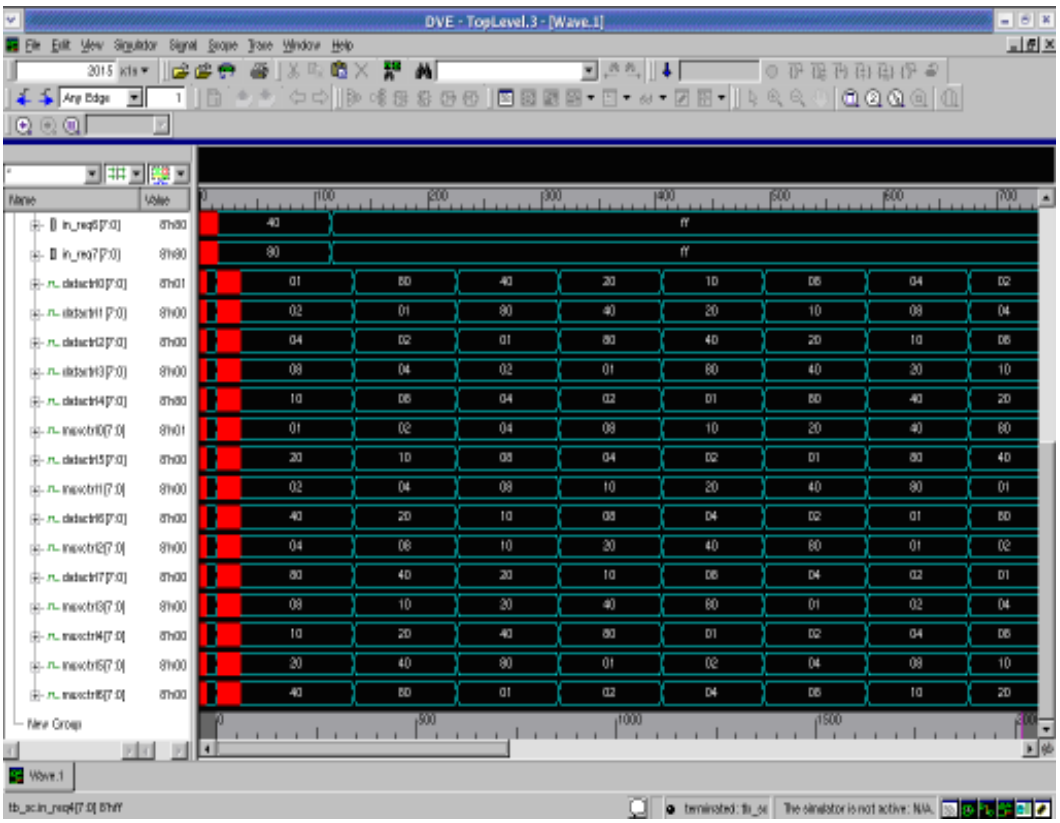
9. Simulation of PPE using XILINX ISE 9.1:



10. Simulation result of Arbiter using XILINX ISE 9.1:



11. Simulation result of Scheduler using XILINX ISE 9.1:

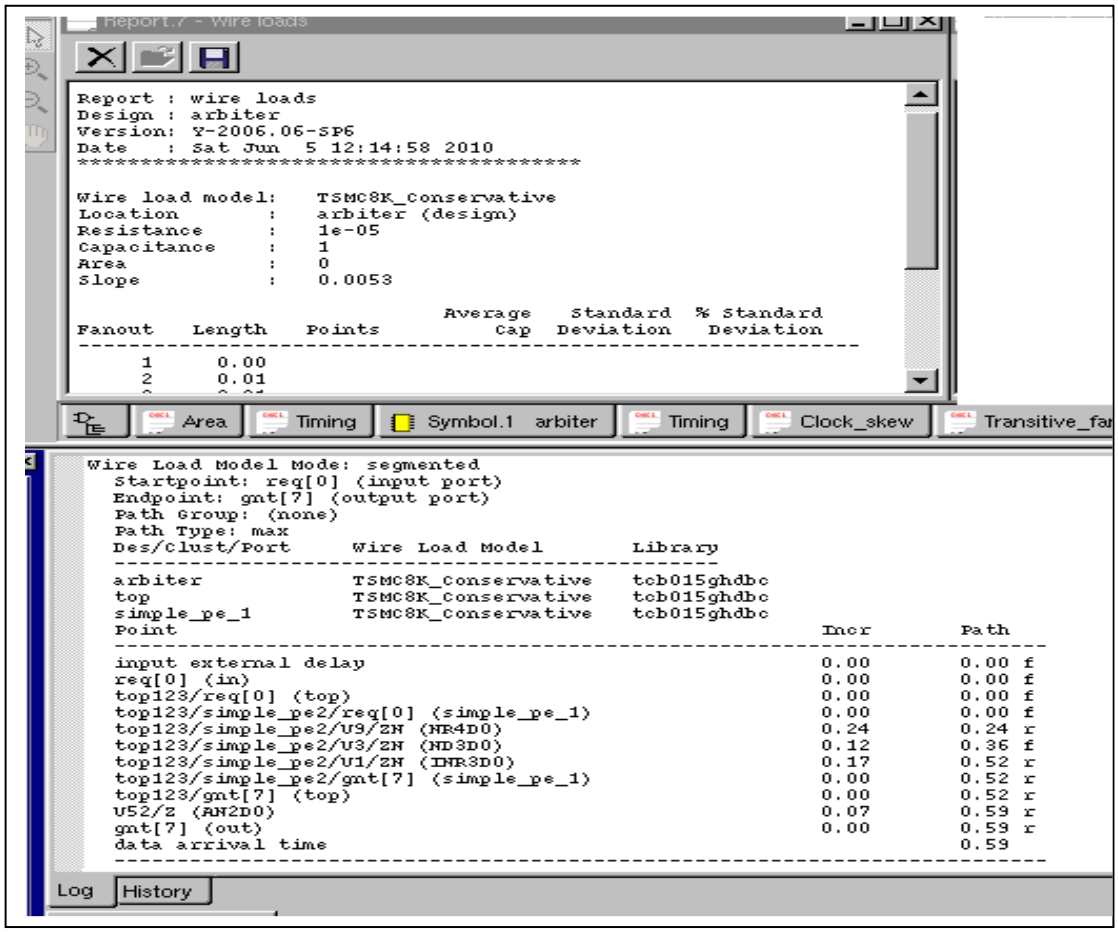


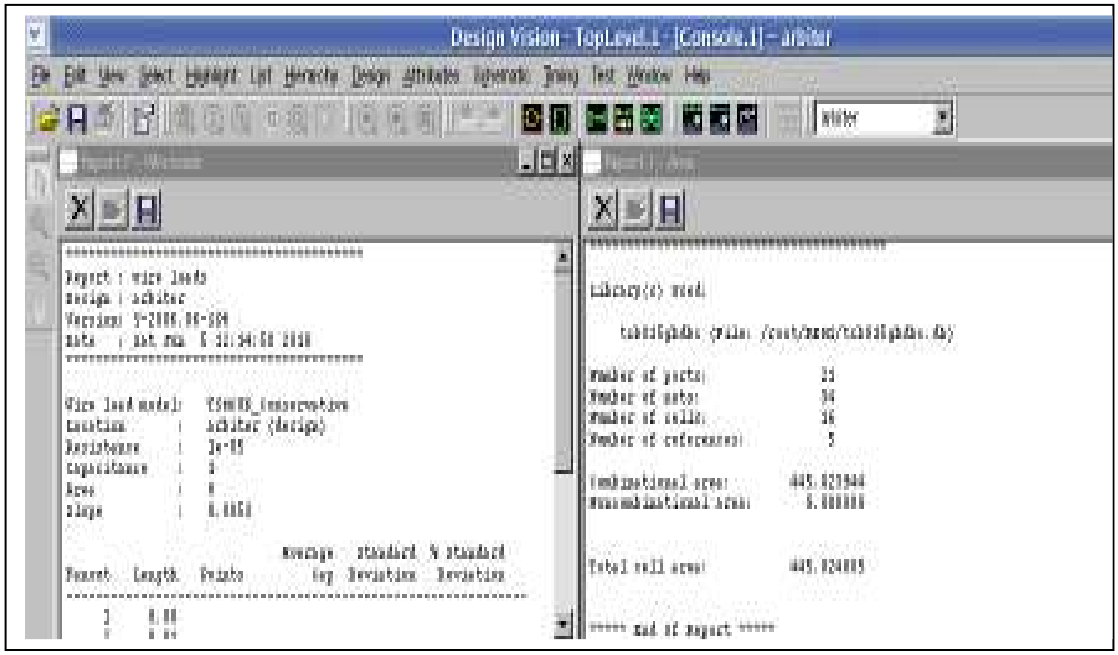
12. Area and Timing result of scheduler:

```
****************************************
Report : area
Design : sc
Version: Y-2006.06-SP6
Date   : Wed Aug  4 16:09:41 2010
****************************************

Library(s) Used:

    tcb015ghdbc (File: /home/student1/aug_03/tcb015ghdbc.db)

Number of ports:              220
Number of nets:               877
Number of cells:              365
Number of references:          47

Combinational area:       16194.731445
Noncombinational area:    4199.036621
Net Interconnect area:      undefined  (Wire load has zero net
area)

Total cell area:          20393.855469
Total area:                 undefined

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : sc
Version: Y-2006.06-SP6
Date   : Wed Aug  4 16:10:27 2010
****************************************

Operating Conditions: BCCOM   Library: tcb015ghdbc
Wire Load Model Mode: segmented

  Startpoint: datactrl4_reg[7]
              (rising edge-triggered flip-flop)
  Endpoint: in_dec_valid[4]
            (output port)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
```

```
arbiter_15          TSMC8K_Conservative   tcb015ghdbc
arbiter_14          TSMC8K_Conservative   tcb015ghdbc
arbiter_13          TSMC8K_Conservative   tcb015ghdbc
arbiter_12          TSMC8K_Conservative   tcb015ghdbc
arbiter_11          TSMC8K_Conservative   tcb015ghdbc
arbiter_10          TSMC8K_Conservative   tcb015ghdbc
arbiter_9           TSMC8K_Conservative   tcb015ghdbc
arbiter_8           TSMC8K_Conservative   tcb015ghdbc
arbiter_7           TSMC8K_Conservative   tcb015ghdbc
arbiter_6           TSMC8K_Conservative   tcb015ghdbc
arbiter_5           TSMC8K_Conservative   tcb015ghdbc
arbiter_4           TSMC8K_Conservative   tcb015ghdbc
arbiter_3           TSMC8K_Conservative   tcb015ghdbc
arbiter_2           TSMC8K_Conservative   tcb015ghdbc
arbiter_1           TSMC8K_Conservative   tcb015ghdbc
sc                  TSMC8K_Conservative   tcb015ghdbc
arbiter_0           TSMC8K_Conservative   tcb015ghdbc
req_2               TSMC8K_Conservative   tcb015ghdbc

Point                                     Incr       Path
-----------------------------------------------------------
datactrl4_reg[7]/CP (DFD1)                0.00       0.00 r
datactrl4_reg[7]/Q (DFD1)                 0.18       0.18 f
U442/ZN (NR4D0)                           0.17       0.35 r
U441/ZN (CKND2D0)                         0.09       0.43 f
U361/ZN (INR2D0)                          0.08       0.52 f
in_dec_valid[4] (out)                     0.00       0.52 f
data arrival time                                    0.52
-----------------------------------------------------------
(Path is unconstrained)
```

10. Experiment results of Arbiter using SYNOPSYS Design Vision





11. 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] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M.Horowitz, "*The Tiny Tera: A Packet Switch Core*," IEEE Micro, vol. 17, pp.26-33, Jan.-Feb. 1997.
[3] Wijetunga, P., "High-performance crossbar design for system-on-chip," *System on-Chip for Real-Time Applications, 2003. Proceedings. The 3rd IEEE International Workshop on* , vol., no.pp. 138- 143, 30 June-2 July 2003
[4] Pape, J; "Implementation of an On-chip Interconnect Using the i-SLIP Scheduling Algorithm: Intermediate Report – Specification and Timeline," Nov 2006.
[5] Gupta, P.; McKeown, N., "Designing and implementing a fast crossbar scheduler," *Micro, IEEE* , vol.19, no.1pp.20-28, Jan/Feb 1999.
[6] Mhamdi, L., Kachris, C., and Vassiliadis, S. 2006. A reconfigurable hardware based embedded scheduler for buffered crossbar switches. In *Proceedings of the 2006*
ACM/SIGDA 14th international Symposium on Field Programmable Gate Arrays (Monterey, California, USA, February 22 - 24, 2006). FPGA '06. ACM Press, New York, NY, 143-149.
[7] S.Q. Zheng, M. Yang, and F. Masetti-Placci, "Constructing schedulers for highspeed, high-capacity switches/routers," Int. J. Comput. Appl., vol.25, no.4, pp.264–271, 2003.
[8] SYNOPSYS Manual for Design Vision/Design Compiler available at SOLVNET{https://solvent.synopsys.com/}.

Authors

Authors' profile

Vilas Nitnaware pursuing PhD from R.S.T.M.Nagpur University, India, in the field of "SOC Interconnect". He is an Assistant Professor and working as a coordinator for the department of Electronic Design Technology since last 10 years. He has been associating with Microprocessors and Microcontrollers related subjects right from his inception. His masters was in the field of VLSI. He undergone training on SYNOPSYS "Design Compiler". He developed some Embedded System products and also delivered training on microcontroller and its applications at various Institutes.

Dr. Shyam S. Limaye received his PhD from Nagpur University in the faculty of Electronics Engg. Currently he is working with JIT as a Principal. He has got vast experience of 33 years in teaching as well as in industry. He also carried out no. of consultancy projects at DCM Data products, Delhi, PSI Data systems Bangalore, Zen and Art New York. His area of specialization includes Digital Signal processing, VLSI design, LDPC codes, CORDIC algorithm. He is recognized Supervisor to guide the PhD scholars in Nagpur University.

.