Folded Architecture of Scheduler for Area Optimization in On-Chip Switch Fabric

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 the feature sizes of the manufacturing processes is constantly shrinking, the possibility and demand for more functionality on a single chip goes up. This can lead to many problems,e.g. as the memory access bandwidth through the bus gets too low to cope with the demand, also the electrical performance of the bus gets degraded as the number of modules are increased. Our proposed architecture makes use of a switch fabric structure to eliminate the traditional drawbacks of bus based design.

Scheduler becomes the integral part of the switch which decides the scheduling of the SOC devices. In this paper, we have proposed an area efficient scheduler which saves around 22 - 26% of the total scheduler area on the silicon die. This becomes possible because the arbiter we designed is capable of executing two different steps of Islip algorithm in two different clock cycles. In the first cycle, it acts as a grant arbiter while the next cycle makes it an accept arbiter. The design is modified using the folding concept which is used to reduce the silicon area by time multiplexing many algorithm operations into a single functional unit. Both the design of the scheduler is synthesized using 90nm SAED library using Design Compiler of SYNOPSYS with the design constraint of input delay, output delay and clock skew. The original scheduler occupies around 22206 area unit while the proposed scheduler occupies around 17285 area unit of the total silicon area considering the constraint of input delay, output delay and clock skew. The area includes both cell area (Combinational + N-Combinational) and Interconnect area.

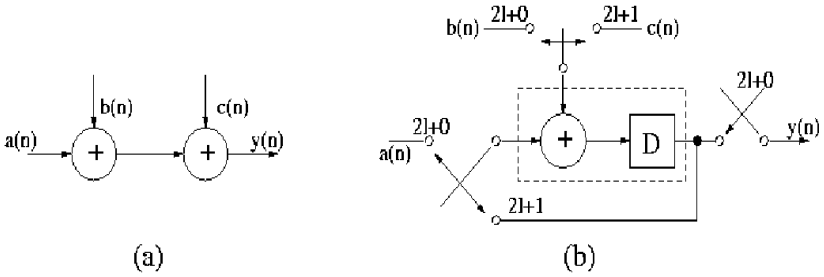
Keywords*: Switch fabric, Folding, Virtual queues, Thermometer encoding, PPE, state pointer.*

1. Folding

It is a technique to reduce the silicon area by time-division multiplexing many algorithm operations into a single functional unit[7]. Figure (a) shows an architecture using two adder units. One output is produced every clock cycle. Figure (b) shows a folded architecture where two additions are folded / time-multiplexed to a single pipelined adder. In this case,

one output is produced every two clock cycles. Therefore input should be valid for two clock cycles.

In general, the data on the input of a folded realization is assumed to be valid for N cycles before changing, where N is the number of algorithm operations executed on a single functional unit in hardware.



2. Crossbar architecture

A crossbar consists of N horizontal buses (rows) and N vertical buses(columns). In our design, we are proposing 8-SOC devices, so the value of N shall be equal to 8. Each horizontal bus is connected to an input port and each vertical bus is connected to an output port. Crossbar switches are fully connected switches. Therefore, in a crossbar switch, there is a direct path from every input to every output. **Figure 1** shows crossbar architecture with input queues.

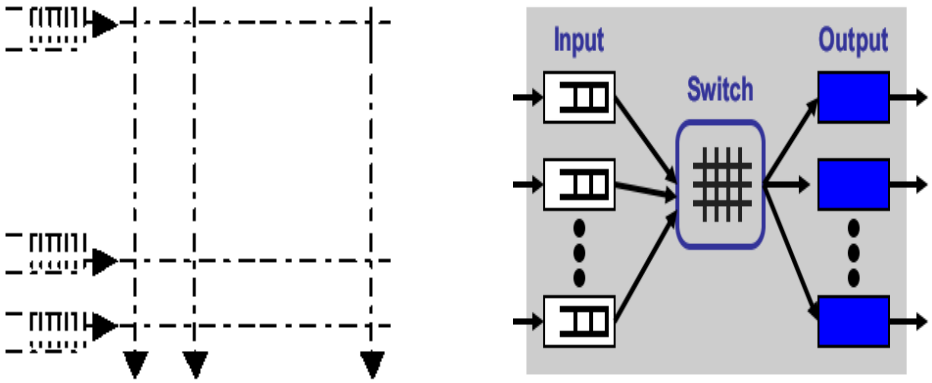


Figure 1: An input queued switch with crossbar architecture

Crossbars provide a direct connection between each input and output port. The speed of the crossbar depends on whether input queues or output queues are used. In case of input queues, the input and output port controllers have the advantage of working with merely the speed of the links. If output queues are utilized, the switch fabric has to be fast enough not to cause contention at the output ports. Crossbar-based systems can be significantly less expensive than bus or ring systems with equivalent performance because the crossbar allows multiple data transfers to take place simultaneously. Furthermore, crossbars are non-blocking, which means any input-output pair can talk to each other as long as they do not interfere with

other input-output pairs. However, in the absence of a fast scheduling algorithm the crossbar becomes a performance bottleneck for big switches. Crossbars are generally expensive, but compared to the total cost of a switch, the crossbar component contributes only a small fraction.

Here, we are using the mux based crossbar to connect the input port to output port once the scheduler declares the verdict. This helps us to make the design fully synthesizable. More efficient designs in the form of sense amps and pass gates are available, but they are not synthesizable.

3. The Scheduling Algorithm: i-SLIP

The proposed scheduler is embodied with i-SLIP algorithm. It has got its own traditional unique characteristics. There is no connection starvation, an output will continue to grant to the highest priority requesting input until it is successful. In one iteration, under heavy load, queues with a common output all have the same throughput. The algorithm converge in N iterations, but simulation suggest that on average, the algorithm converges in fewer than log2*N* iterations.

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

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

Step 2: *Grant.* If 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 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 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.

The graphics shown below in Fig. 2 help to illustrate the algorithm.

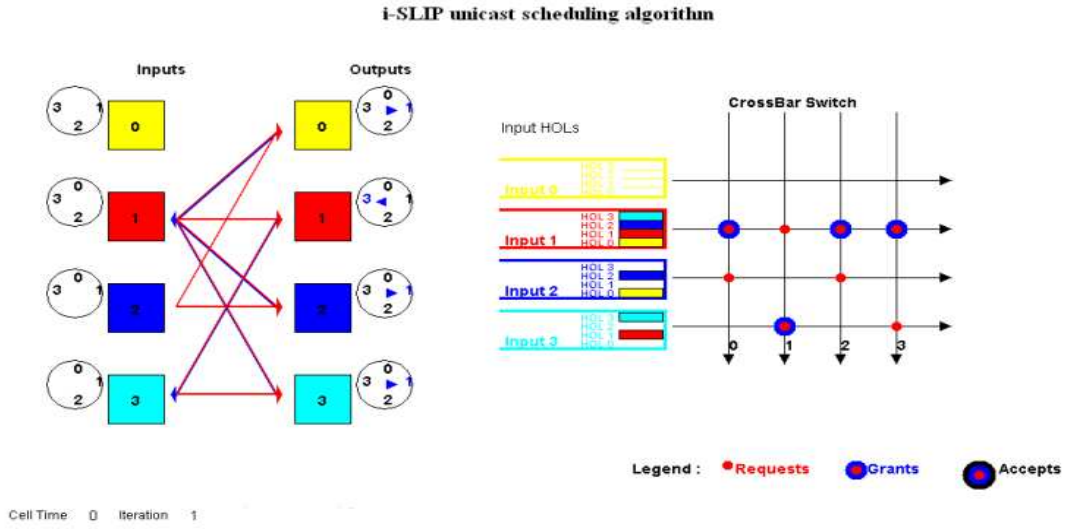


Figure 2: Working of i-slip scheduling algorithm with simple crossbar

4. Arbiters

The arbiters are an important component 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. **Figure 3** illustrates the arbiter design chosen for this implementation. The arbiter[5] 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 **Figure 3**, 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.

It has some state (called round-robin pointer, *P_enc*, of width log_2N 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 a log_2N-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.

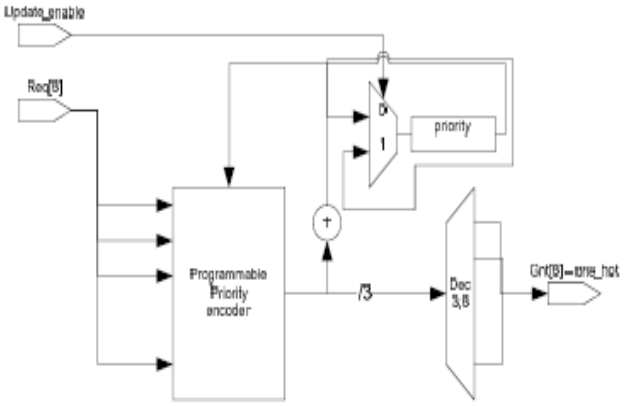


Figure 3: Original Arbiter

5. Proposed Arbiter:

The proposed arbiter as shown in Fig 4 is responsible for performing grant as well as accept function through time division multiplexing. It holds two priority update registers each

for grant[pri_gnt] and accept[pri_acc] arbitration, where original arbiter[6] was holding only one update pointer register as it had to perform only one function. This is the folding concept as it was proposed by[7] where a single component is capable of performing two different operations on time sharing basis. As per the original concept of the algorithm, in one clock cycle, all the three steps were executed - request, grant and accept - as we had exclusive set of arbiters to perform it. While in proposed arbitration, in one clock cycle grant is performed while in the next clock cycle accept is performed, as per the simulation results shown.

During grant mode of operation, this arbiter acts as the grant arbiter. It accepts the request signal for a given output port and generates the grant signal for that output port. During the accept mode of operation, this arbiter accepts the output port grant signal as input and generates the input port accept signal.

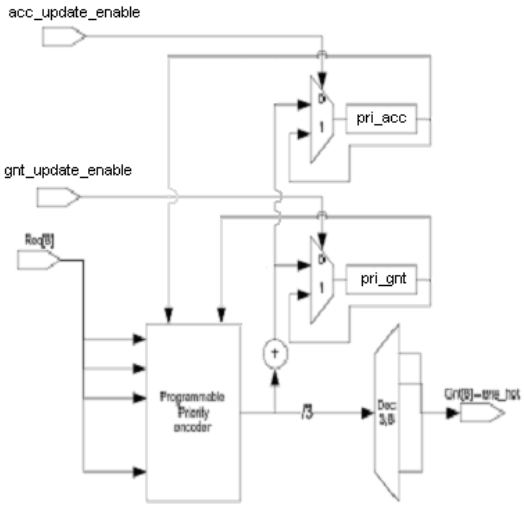


Figure 4: Proposed Arbiter to perform grant as well as accept function

6. Original 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 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.

This algorithm 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[1] 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.

Figure 5 shows the i-SLIP scheduler[10] 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 8 such vectors (1 per crossbar input port), each with 8 bits (1 per destination per input). This 8x8 state is used as the request vector to 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 iteration.

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).

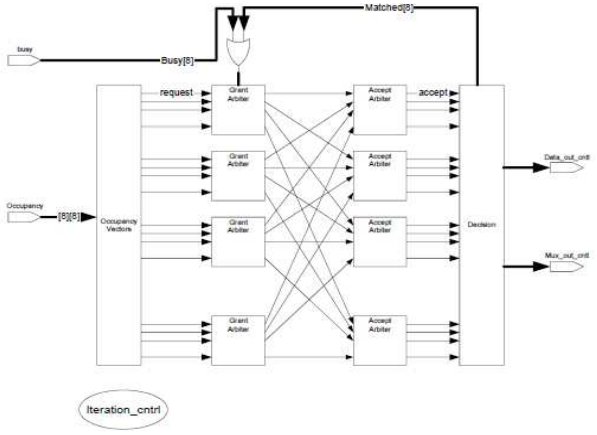


Figure 5: Original scheduler with independent grant and accept funtion.

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.

7. 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, the design chosen here for the original scheduler utilizes a request masking mechanism and two simple non-programmable priority encoders for this implementation.

Simple priority encoder is implemented 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 to achieve a very low delay through the priority encoder[5].

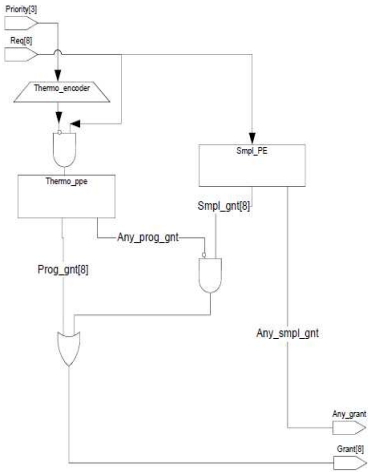


Figure 6: Original PPE in the arbiter

8. Proposed PPE

As shown in Figure 7, only 2:1 mux is added to select the priority for accept and grant arbiter. The three bit priority selected by the mux will be given as an input to the encoder. The encoded bits are the inputs to the AND while the 8-bit request is the other input.

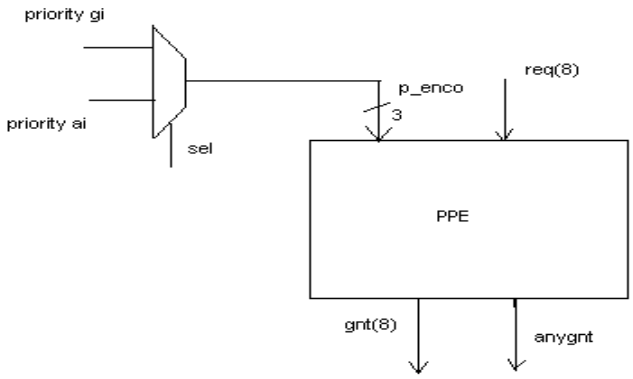


Figure 7: Proposed PPE diagram including gi[pri_gnt] & ai[pri_acc]

The bits will be finally encoded by the thermo_ppe and the 8-bit signal, which is nothing but the programmable grant, will be given as an input to the OR gate. At the same time the request signals will also be given to the simple priority encoder. It will encode the bits and the simple 8 bit grant signal will be anded with another grant signal.

The output of this second and gate is nothing but the programmable grant signal. This signal will be given as another input to the OR gate. Finally the 8-bit grant will be obtained at the output of the programmable priority encoder.

The other hardware required for the arbiter design is encoder and the latch as the feedback blocks.

9. Proposed Scheduler:

As shown in the original scheduler [10], there are total 16 arbiters used for 8 SOC devices. Here the concept of the folding mechanism is exploited to minimize the silicon die area wherein a single component is used to perform two different operations of the same algorithm. Here the single arbiter is performing dual role of grant as well as accept function on time division multiplexing. This is shown in fig.8, where only one set of arbiters is used to execute both the operations, so the number of required arbiters is reduced to only 8. The architecture of the original arbiters and the original PPEs modified are shown in Figures 4 and 7 respectively.

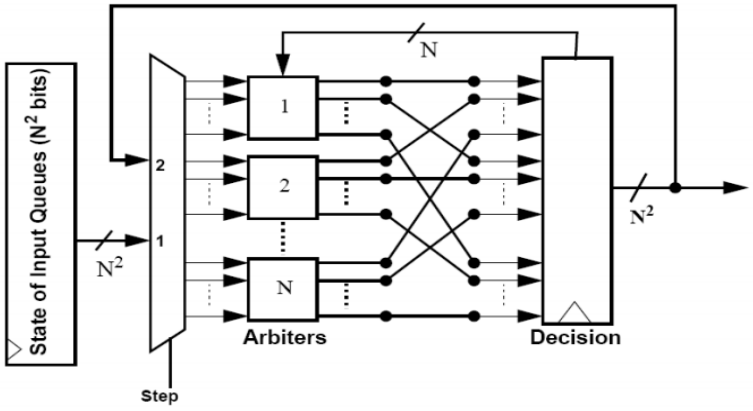


Figure 8: Proposed Scheduler

10. Synthesis Results:

The results shown below are obtained using DC of SYNOPSYS with 90nm saed library. The gate level netlist is obtained without setting any prior constraint. Without constraint the DC optimizes the design for minimal area and assumes ideal clock[11]. As shown in Table1, the original scheduler takes 17129 units of the silicon die while the proposed scheduler requires 12950 units. With the proposed scheduler with folding concept, it is possible to save area upto 25%.

Table 1

Original Scheduler[U]		*Proposed Scheduler[U]*
Report : area Design : sc Version: Y-2006.06-SP6 Date : Tue Oct 19 14:17:25 2010 ********** Library(s) Used: saed90nm_typ (File: /home/student1/2010-09-22/risc_design/libs/saed90nm_typ.db) Number of ports: 220 Number of nets: 851 Number of cells: 379 Number of references: 36 Combinational area: 12996.039062 Noncombinational area: 3035.730469 Net Interconnect area: 1098.012329 Total cell area: 16031.669922 Total area: 17129.681641		Report : area Design : sc Version: Y-2006.06-SP6 Date : Tue Oct 19 14:17:40 2010 ********** Library(s) Used: saed90nm_typ (File: /home/student1/2010-09-22/risc_design/libs/saed90nm_typ.db) Number of ports: 222 Number of nets: 751 Number of cells: 422 Number of references: 30 Combinational area: 8836.385742 Noncombinational area: 3284.561035 Net Interconnect area: 829.488525 Total cell area: 12120.953125 Total area: 12950.441406

As shown in Table 2, the longest combinational path in original scheduler takes around 0.57 ns of time while in proposed scheduler the longest path takes around 0.64ns of time. The time overhead in the proposed scheduler is increased because of the inclusion of the additional update pointer register in the same arbiter.

Table2

Original Scheduler[U]		*Proposed Scheduler[U]*
Report : timing -path full -delay max -max_paths 1 -sort_by group Design : sc Version: Y-2006.06-SP6 Date : Tue Oct 19 14:12:14 2010 ********** Operating Conditions: TYPICAL Library: saed90nm_typ Wire Load Model Mode: enclosed Startpoint: datactrl4_reg[4] (rising edge-triggered flip-flop) Endpoint: in_dec_valid[4] (output port) Path Type: max data arrival time 0.57 --- (Path is unconstrained)		Report : timing -path full -delay max -max_paths 1 -sort_by group Design : sc Version: Y-2006.06-SP6 Date : Tue Oct 19 14:18:47 2010 ********** Operating Conditions: TYPICAL Library: saed90nm_typ Wire Load Model Mode: enclosed Startpoint: datactrl4_reg[4] (rising edge-triggered flip-flop) Endpoint: in_dec_valid[4] (output port) Path Type: max data arrival time 0.64 --- (Path is unconstrained)

Another set of results are obtained by applying constraint of input delay, output delay and clock skew as per the guidelines by SYNOPSYS manual[11]. If no prior set of input and output delays are mentioned in the specification, then we have to consider its values as 40% of the clock period. First set of readings in both Table3 and Table4 are the outcome of the first consideration while the second set of readings stand for fixed output and input delays. In both set of readings, the common point of discussion is area with 20-25% improvement over the original scheduler.

Proposed Scheduler with constraint of input delay, output delay and clock skew:

Table 3

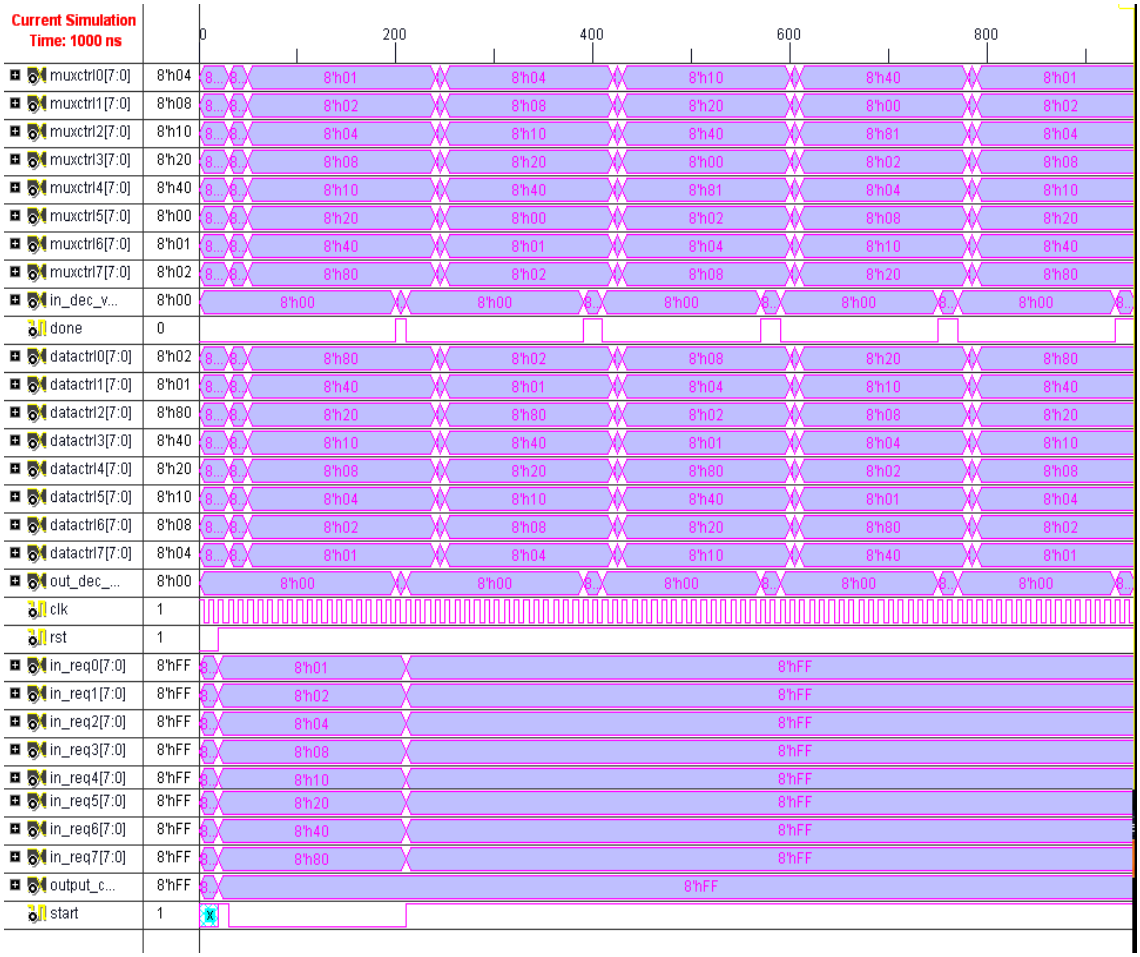
Total Area	*Clock Period*	*input delay*	*output delay*	*status of slack*
17285	30ns	18ns	18ns	met
15524	30ns	2ns	0.5ns	met

Original Scheduler with constraint of input delay, output delay and clock skew:

Table 4

Total Area	*Clock Period*	*input delay*	*output delay*	*status of slack*
22206	4ns	2.4ns	2.4ns	met
21504	4ns	2ns	0.5ns	met

11. Simulation result of folded Scheduler using XILINX ISE:



12. Conclusion

Since the majority of the area in the scheduler is consumed by the grant and accept arbiters, the structure of the arbiter in Figure 3 is modified to Figure 4 to reuse the priority encoding logic for both the Grant and Accept arbiters. This saves area, but introduces additional sequential logic overhead as the Grant result must be stored in the decision registers on every other cycle. The different test patterns are applied to the scheduler to test its functionality. The algorithm works with its original nature except in every alternate iteration, it takes only two clock cycles to complete its scheduling as shown in the simulation result.

The proposed scheduler is synthesized using Design Compiler of SYNOPSYS in 90nm SAED synthesis library. In the mode of synthesis, the design was optimized using input and output delays at 40% of clock period as it is recommended in SYNOPSYS manual. The most optimized results are available at 30ns with the total area of 17285 units while with lowering the values of input/output delays as shown in Table 3, the area is further reduced to 15524 units. The corresponding area readings of the original scheduler is higher at least by 22-25% as per the readings in Table 3 and Table 4.

13. 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] Nitnaware V.; Limaye S. S., "An efficient arbiter in the On-chip scheduler embodying i-SLIP algorithm ", WASET 2010, Amsterdam, September 28th - 30th.

[7] A Book by Keshabh K. Parhi on "VLSI Signal Processing"

[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 high speed, high-capacity switches/routers," Int. J. Comput. Appl., vol.25, no.4, pp.264–271, 2003.

[8] R. Ahuja et al. "Multicast Scheduling for Input-Queued Switches," *IEEE JSAC*, June 1997.

[9] A. Dua, N. Bambos, W. Olesinski, H. Eberle, N. Gura: "Backlog aware low complexity schedulers for input queued packet switches", *IEEE Symp. on High-Perf. Interconnects*, 2007.

[10] Nitnaware V., Limaye S. S., "Time Efficient Arbiter in the design of Scheduler embodying Islip algorithm for on chip interconnection," Int. J. IJAST, vol.21, ISSN: 2005-4238, August 2010.

[11] SYNOPSYS Manual for Design Vision/Design Compiler available at SOLVNET {https://solvent.synopsys.com/}.

Authors

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.