International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 2 Issue 9, September-2013, pp: (1-9), Available online at: **www.erpublications.com**

Speed efficient implementation of round robin arbiter design using VERILOG

Ruma Deb, Dr Rajrajan
Dept. of Electronics & Communication Engg., Sathyabama University, Chennai (TN), India

Abstract: Round robin arbitration is commonly used for scheduling. With the revolutionary improvement in optical and electronics interconnection technologies, a very fast arbiter design is required to match the speed of high performance buses. This project present the design of high speed PPE arbiter the fairness of the arbiter is evaluated and possible enhancement to the arbiter. We show that our design is faster than existing round robin arbiter design. The main contribution of this project is the design of fast round robin arbiters. 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 PPE arbiter design for each cycle, one of the masters (in round-robin order) has the highest priority (i.e., owns the token) for access to a shared resource. If the token-holding master does not need the resource in this cycle, the master with the next highest priority who sends a request can be granted the resource, and the highest priority master then passes the token to the next master in round-robin order.

INTRODUCTION

The arbitration plays a critical role in determine performance of bus based system, as it assign priorities with which processors is grant the access to the shared communication resources Arbiters exist in nearly every logic design. Many systems exist in which a large number of requesters must access a common resource. The common resource may be a shared memory, a networking switch fabric, a specialized state machine, or a complex computational element. An arbiter is required to determine how the resource is shared amongst the many requesters. When putting an arbiter into a design, many factors must be considered. The interface between the requesters and the arbiter must be appropriate for the size and speed of the arbiter.

PREVIOUS WORK

One common arbitration scheme is the simple priority arbiter. Each requester is assigned a fixed priority, and the grant is given to the active requester with the highest priority. For example, if the request vector into the arbiter is request [N-1:0], request [0] is typically declared the highest priority. If request [0] is active, it gets the grant. If not, and request [1] is active, grant [1] is asserted, and so on. Simple priority arbiters are very common when hosing between just a few requesters. For example, a maintenance port may always be lower priority than the functional port. ECC corrections may always be higher priority than all other requests. Priority arbiters are also often used as the basis for other types of arbiters. A more complex arbiter may reorder the incoming requests into the desired priority, run these scrambled requests through a simple priority arbiter, then unscramble the grants which come out.

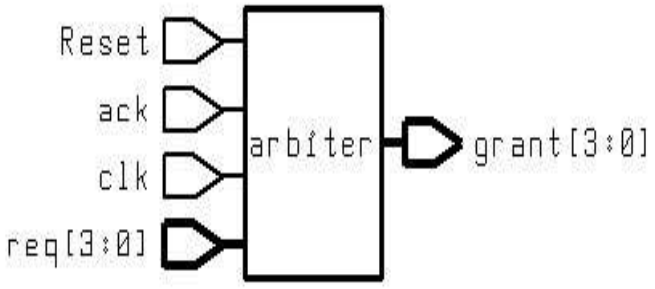


Fig 1: Block Diagram for Four Bus Masters.

PROPOSED WORK

The need for efficient implementation of arbiter has increased in the recent years due to the advent of on-chip interconnection networks that require low latency message delivery. The core function of any arbiter (scheduler) is arbitration that resolves conflicting requests for the same output. Since,the arbiters directly determines the operation speed of the scheduler, the design of faster arbiters is of paramount importance. In this project, we present a new bit-level algorithm and new circuit techniques for the design of programmable priority arbiters that offer. From the experimental results it is derived that the proposed circuits are more than 25% faster than the most efficient previous implementations. What we think are the main contributions of this project:

i) Request-grant and accept phases can be pipelined across different iterations to save precious clock cycles.
ii)Programmable priority encoders can be speeded up to be almost exactly as fast as static priority encoders, resulting in fast arbiters like ultra-high- speed switches significantly more efficient implementations compared to already-known solutions.

In my proposed work implemented PPE arbiter to speed up the arbiter, has no fixed priority. An extra port pointer is used. In this at single clock pulse it will check for any user asking for request if any, it assign the grant to it. The pointer moves in a round that is cyclic order as shown in above figure 4. A round-robin arbiter is used to resolve conflicting requests generated from various sources for a shared resource in a directional and cyclic order.

Requests are generally tabulated by a fixed order in a multiple bit register with each bit corresponding to a specific request. An asserted bit meaning a request is issued from the source. At most one request can be granted at any time – this is defined as a turn being given. If the turn is given in a circular manner, it can be viewed as an exclusive token cycling around. If the request is in the same position as of the token being asserted, it is called a turn-hit, otherwise it is a turn-miss. Fig.2 illustrates both the hit and the miss cases for an arbiter managing six candidates. The token is cycling around by shifting to the right each cycle. It gives turn to the second candidate but that one does not request so the turn is missed. Although along the shifting direction there are asserted requesting candidates, the nearest one is the fourth one, it must wait until its turn after two cycles later assuming the request still persist. Apparently there are much more cycles wasted in an arbiter managing more candidates especially when they generate requests very unevenly. The arbiter generates the acknowledgement or grant signal if both of the following conditions are satisfied.

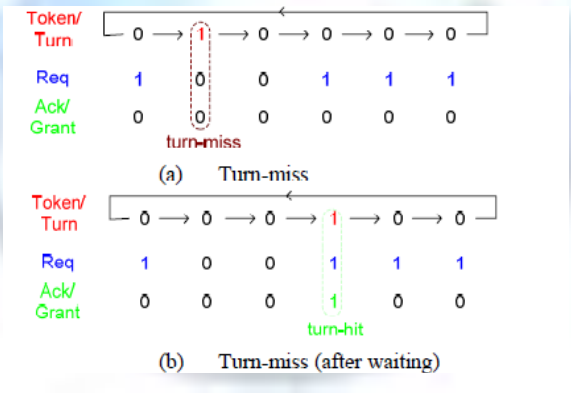


Fig 2: Trun Miss And Turn Hit In Round Robin Arbiter

- There is really a request, or the request bit is asserted;
- The turn is exclusively given to this request, or a turn-hit.

Although the design of the arbiter in this way is quite straight forward and fast, it may potentially degrade system performance in which existing requests for service are denied or delayed for a long time and may force the candidate to switch request for other possible resources (maybe already busy) – if not, the system basically denying service while the requested resource is available. If the arbiter is designed in such a way that it can skip the non-requesting candidate and find and grant the closest asserted candidate along its searching direction, it can improve overall system performance. Therefore in my project it will check only those requestor which is asking for request in cyclic manner. If suppose there are nine user or requestor and pointer moving in cyclic order. Suppose request [9] ask for grant and at the same time request [2] and the pointer is at request [2] so it will assign the grant or acknowledgement. On next clock cycle if only request [9] asking request it will assign the grant to it.

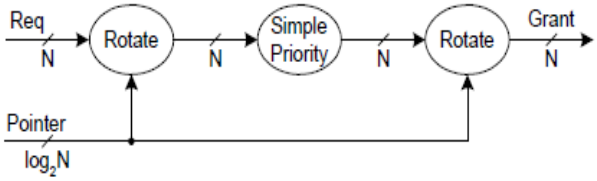


Fig 3: Block Diagram Of Pointer Update

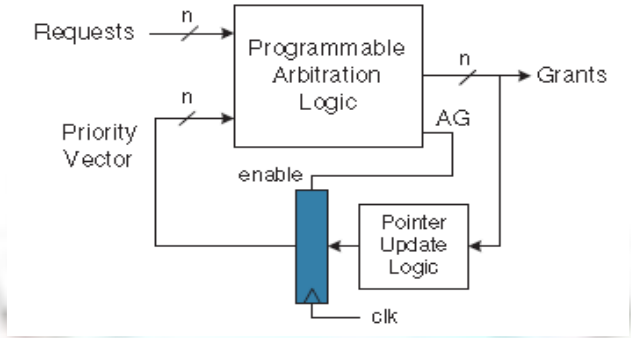


Fig 4: Block Diagram Of Progrmmable PRIORITY ENCODER

The key shortcoming of priority arbiters is that, in very busy systems, there is no limit to how long a lower priority request may need to wait until it receives a grant. A round-robin arbiter on the other hand allows every requester to take a turn in order. A pointer register is maintained which points to the requester who is next. If that requester is active, it gets the grant. If not, the pointer is then moved to the next requester. In this way, the maximum amount of time that a requester will wait is limited by the number of requesters. In this project increase in speed is obtained by PPE arbiter design using Slip algorithm.

SLIP ALGORITHM

The SLIP algorithm - a novel algorithm for scheduling cells input-queued switches. The SLIP algorithm uses rotating priority ("round-robin") arbitration to schedule each active input and output in turn. The main characteristic of SLIP is its simplicity: it is readily implemented in hardware and can operate at high speed. Before describing SLIP, we begin this chapter with a description of the basic round-robin matching (RRM) algorithm. We show that RRM performs poorly and demonstrate this with some examples. The performance of SLIP for uniform traffic is surprisingly good; in fact, for uniform i.i.d. Bernoulli arrivals, SLIP with a single iteration is stable for any admissible load. The arbiters in SLIP have a tendency to desynchronize with respect to one another. The operation in which SLIP behaves non-monotonically: increasing offered load can actually decrease the average queuing delay.

The i-SLIP algorithm is derived from the SLIP algorithm, which is an improvement upon the Round-Robin Matching algorithm. To alleviate head of lock blocking at input queus, each input maintain separate queues 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 thisiteration, several possible input and output ports may remain unutilized. The i-SLIP algorithm uses multiple iterations to find paths to utilize as many inputs and output ports as possible. The algorithm leads to the following properties of SLIP.

PROPERTIES OF SLIP ALGORITHM

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). If input i successfully connects to output j, both ai and gj are updated and the connection from input i to output j becomes the lowest priority connection in the next cell time.

Property 2

No connection is starved. 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, a requesting input is always served in less than N2 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.

THREE STEP OF ARBITRATION

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 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 to one location beyond the accepted output

The requester that is pointed to by the round-robin pointer is shifted to the highest priority position, and the other requests are rotated in behind it. This rotated request vector is then sent. The grant vector from the priority arbiter is then "unrotated" to come up with the round-robin arbiter's final grant signal. This is shown in the block diagram.

A round-robin token passing bus or switch arbiter guarantees fairness (no starvation) among masters and allows any unused time slot to be allocated to a master whose round-robin turn is later but who is ready now. A reliable prediction of the worst-case wait time is another advantage of the round-robin protocol. The worst-case wait time is proportional to number of requestors minus one. The protocol of a round-robin token passing bus or switch arbiter works as follows. In each cycle, one of the masters (in round-robin order) has the highest priority (i.e., owns the token) for access to a shared resource. If the token-holding master does not need the resource in this cycle, the master with the next highest priority who sends a request can be granted the resource, and the highest priority master then passes the token to the next master in round-robin order**.**

A round-robin token passing bus or switch arbiter guarantees fairness (no starvation) among masters and allows any unused time slot to be allocated to a master whose round-robin turn is later but who is ready now. A reliable prediction of the worst-case wait time is another advantage of the round-robin protocol. The worst-case wait time is proportional to number of requestors minus one. The protocol of a round-robin token passing bus or switch arbiter works as follows.

In each cycle, one of the masters (in round-robin order) has the highest priority (i.e., owns the token) for access to a shared resource. If the token-holding master does not need the resource in this cycle, the master with the next highest priority who sends a request can be granted the resource, and the highest priority master then passes the token to the next master in round-robin order. In fig 4.4 shows as ASM chat the working of proposed arbiter design. In this at single clock cycle it checks for a request and assigns the grant to it and update the pointer. The pointer move in cyclic order as explained earlier.

In the state diagram of the new arbiter the pointer is the extra port which is used. In single clock pulse pointer checks which requestor or user is asking for request it provide with the grant and update the pointer. Different cases is discusses for explaining the different condition of the arbiter.

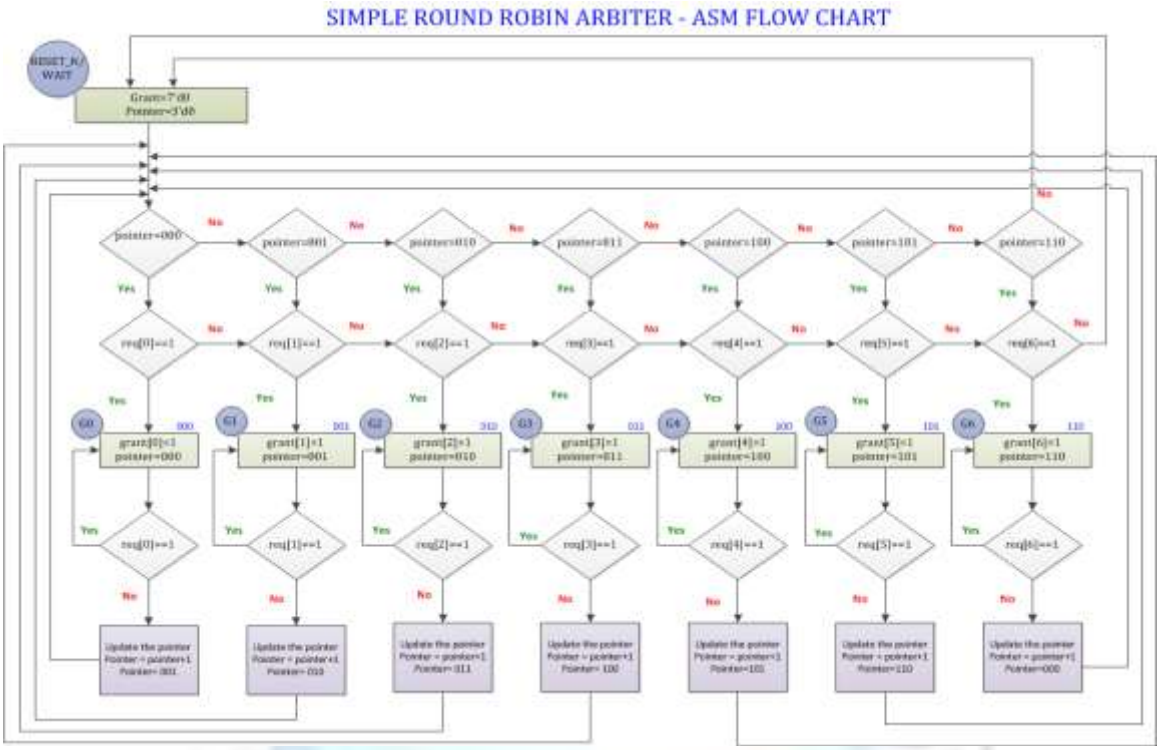


Fig 5 : simple round robin arbiter – asm flowchart

Case 1:

"none of the request" from source units, pointer position reset to "000", next case priority is coming into the picture, arbiter issue the grant to the higher priority request.

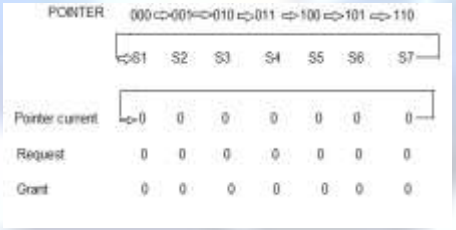


Fig 6: Conditions With No Request

Case 2:

Based on the case1- arbiter position "000", there is "no request" from S1 unit; the next active requester gets the grant on same clock cycle. i.e. arbiter issue the grant to the 'S2' unit so long as the request is present, once the arbiter issue the grant- pointer updated by "001", pointer maintain the same position so long the current S2 request is present, once the current request going to inactive the pointer get increment by 1 i.e. "010"

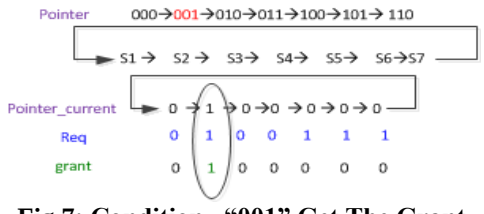


Fig 7: Condition "001" Get The Grant

Case 3:

Based on the case2 -once the 'TD' request is inactive, pointer increment by '1' i.e pointer position "010" and also it have the request, arbiter issue the grant to the 'SRIO' unit so long as the request is present, pointer maintain the same position so long the (SRIO) request present, once the current request going to inactive, then pointer get increment by 1 i.e. "011"

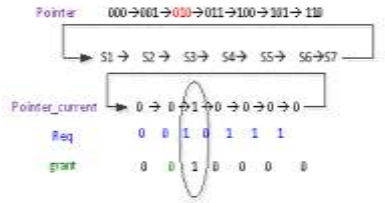


Fig 8: Condition When More Than One User Need The Grant

Case 4:

Based on the previous (pointer – "010") case, once the previous case request in-active , pointer is increment by 1 i.e. pointer "011", but it not have request , so next active request gets the grant from arbiter on the same clock cycle and pointer also updated to "100" , it have the request, arbiter issue the grant to the S5 unit so long as the request is present, pointer maintain the same position so long the current S5request is present, once the current request is going to inactive the pointer get increment by 1.

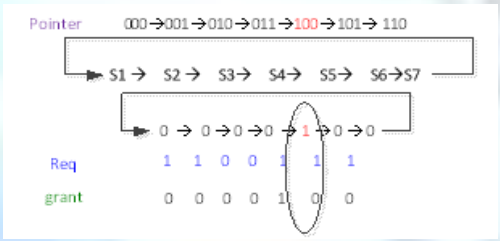


Fig 9: Condition many user need the grant.

This PPE takes n 1-bit-wide requests and the log n-bit-wide pointer (P_enc) as inputs. It then chooses the first nonzero request value beyond (and including) Req [P_enc], resulting in an n-bit grant. Clearly, this combinational block carries out the core function of contention resolution. 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 design of a fast round-robin arbiter reduces to designing a fast PPE. A fast PPE would work in any arbiter, regardless of the pointer update mechanism. The design of a fast, nonprogrammable priority encoder (smpl_PE) is well known. It is the presence of programmability that complicates the design of our PPE. It uses n duplicate copies of the simple priority encoder, smpl_PE, and uses the programmable priority input to select which priority encoder to use. In this with ptr as the select signal. The PPE could equivalently obtain any grant by a simple n-bit OR of the Req inputs. This design is reasonably fast for small n. Clearly, this design requires a large area of silicon, and it is unsuitable for a large n. Although the timing should scale well, area consumption grows geometrically for increasing values of n.

Fig10: programmable high speed input output arbiter.

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. Some require rotating the request vector and then rotating the resulting priority vector, while others resemble a ripple-carry adder design. However in my design I implemented with rotating the request vector and then rotatating the resulting priority vector.

SIMULATION RESULT

The aim of the project is to increase the speed of round robin arbiter by increasing frequency and decreasing the delay. Programmable priority encoder based Round Robin Arbiter has been used as a fair (non starvation) scheduling policy in many computer application. This project present novel design of round robin arbiter without any misses – It always grants an available resource to one of legitimate requests, which may be very unevenly generated from various sources. The design is modeled in HDL, using Xilinx ISE 9.1 with SPARTAN 3E family , XC3S250E device , tq144 package logically verified and then synthesized in XST synthesis tool and simulation in modelsim 6.3. The speed of this arbiter with nine users, compared with seven user or requestor design, make it promise well for performance improvement in systems with potential non-uniform requests. The comparison is shown in table 4.1 the frequency with seven users in static priority arbiter is 65.703 MHz and 4.945 delay ns whereas the frequency with nine users in PPE arbiter is 76.488MHz. and delay 4.863 ns thus frequency increased by 10.785 MHz and delay decrease by 0.092ns.

No: of users	Condition	Frequency (MHz)	Delay (ns)
Seven	Existing	65.703	4.945
Nine	Proposed	76.488	4.863
		10.785 increased.	0.092 Decreased.

In the above fig the input ports are reset, clock and vector input i.e request [8:0] and vector output [8:0] i.e. grant.

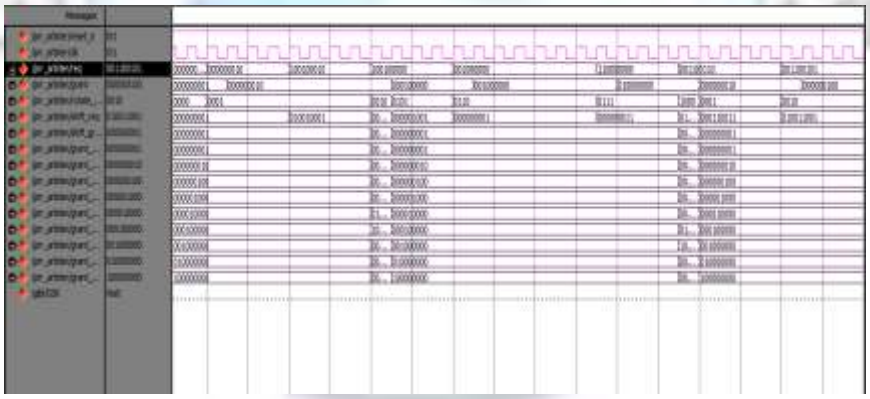


Fig11: simulation result of ppe using xilinx 9.1

Simulation results show that our arbiter leads to speed improvement when compared with existing designs.

RTL VIEW

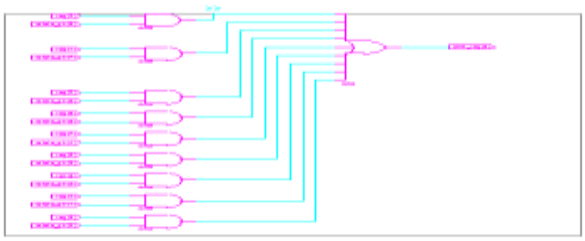


Fig 12 : RTL view

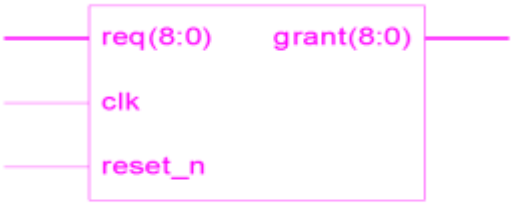
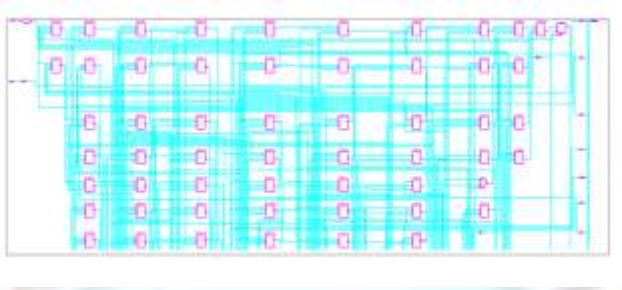


Fig 13: TECHNOLOGY VIEW



SYNTHSIS REPORT

Device utilization summary:

Selected Device: 3s250etq144-4

Number of Slices:	84 out of	2448	3%
Number of Slice Flip Flops:	9 out of	4896	0%
Number of 4 input LUTs:	163 out of	4896	3%
Number of IOs:	20		
Number of bonded IOBs:	20 out of	108	18%
Number of GCLKs:	1 out of	24	4%

TIMING REPORT

Speed Grade: -4

Minimum period: 13.074ns (Maximum Frequency: 76.488MHz)
Minimum input arrival time before clock: 13.895ns
Maximum output required time after clock: 4.863ns
Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

Total memory usage is 190712 kilobytes

PERFORMANCE ANALYSIS

Three basic steps are involved in the design of an arbiter: modeling, logic verification, and synthesis. The design is first modeled and may be represented in schematic graphs. The model is then described in Verilog HDL and the logic can then be verified by using simulation tool modelsim 6.3 version. Finally, the design can be mapped, to be synthesized and reduce delay and increase clock frequency. The synthesis tool used is XST synthesis, which optimizes a design constrained by

timing. The same design can achieve a higher clock frequency. When comparing different design strategies of the same logic implementation, timing is used as the primary target in this project. As we see from comparison table the speed of the proposed work is being increased by 10.785 MHz in frequency and decrease in delay by 0.092ns

CONCLUSION

The project on efficient speed implementation of round robin arbiter using verilog is been achieved. PPE arbiter design using Islip algorithm which has much better performance than other practical algorithms under variant traffic models, without adding much complexity. Islip algorithm meets the criterion of a good scheduling algorithm good performance, fast and simple to implement. PPE arbiter design increases the system speed. It's a much better approach than the static priority arbiter design. The project is been done using Xilinx ISE 9.1 with SPARTAN 3E family , XC3S250E device,tq144 package logically verified and then synthesized in XST synthesis tool and simulation in modelsim.6.3.using verilog HDL. The result obtained is increase in frequency by 10.78 MHz and delay is decrease by 0.092 ns. Thus the aim of the project is being achieved. The frequency of the Round Robin arbiter can further be increased if more better synthesis tool and simulation tool is being used such Synopsys.

REFERENCES

[1]. Ge, J., "Cost-effective Buffered Wormhole Routing". Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, Vol. 3358, pp666-676, 2004.
[2]. Palnitkar, S., "Verilog HDL: A Guide to Digital Design and Synthesis". SunSoft Press, Palo Alto, CA, 1996.
[3]. Nabeel Al-saber, Saurab Oberoi, Roberto Rojas-Cessa andSotirios G. Ziavras, "Concatenating Packets for Variable-Length Input-Queue Packet Switches with Cell-Basedand Packet-Based Scheduling," Proc. IEEE Sarnoff Symposium, Princeton, NJ, April 28-30, 2008.
[4]. M.J. Karol, M. G. Hluchyj, and S.P. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," IEEE Transactions on Communications, 35:1347-56, 1997.
[5]. T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," ACM Trans. Comput. Syst., pp. 319-52, Nov. 1993.
[6]. M. A. Marsan, A. Bianco, E. Leonardi, and L. Milia, "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches," IEEE Transactions on Communications, 47: 1921-33, Dec.1999.
[7]. N. Mckeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," IEEE Transactions on Communications, 47: 1260-67, Aug. 1999.
[8]. A. Mekkittikul, and N. Mckeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," IEEE INFOCOM 98, San Francisco, April 1998.
[9]. N. McKeown, "Scheduling Cells in an Input-Queued Switch," PhD thesis, University of California at Berkeley, May 1995.
[10]. N. Mckeown, and T. E. Anderson, "A Quantitative Comparison of Iterative Scheduling Algorithms for Input-Queued Switches," Computer Networks and ISDN systems, vol.30, pp. 2309-2326, 1997.
[11]. H. J. Chao, and J.-S. Park, "Centralized Contention Resolution Schemes for A Large-Capacity Optical ATM Switch," Proc. IEEE ATM Workshop, Fairfax, VA, May 1998.
[12]. D. N. Serpanos, and P. I. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues," PROC. IEEE INFOCOM 2000, pp. 548-555. 2000.
[13]. R. E. Tarjan, "Data Structures and Network Algorithms," Society for Industrial and Applied Mathematics, Pennsylvania, Nov. 1983.
[14]. N. McKeown, M. Izzard, A. Mekkittikul, and M. Horowitz, "The Tiny Tera: A Small High-Bandwidth Packet SwitchCore," IEEE Micro Magazine, vol.17, No. 1, pp. 26-33, Jan-Feb, 1997.
[15]. Y. Li, S. Panwar, H.J. Chao, "The Dual Round-robin Matching Switch with Exhaustive Service," IEEE HPSR 2002, pp. 58-63, 2002.
[16]. L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches", IEEE INFOCOM 98, pp. 533-539, 1998.
[17]. P. Giaccone, B. Prabhakar, D. Shah, "Randomized scheduling algorithms for high-aggregate bandwidth switches", IEEE J. Sel. Area Commun. 21 (2003), pp. 546-559, 2003.
[18]. R. Rojas-Cessa and C-B. Lin, "Captured-Frame Eligibility and Round robin Matching for Input-queue Packet Switches," IEEE Commun. Letters, vol.8, issue 9, pp. 585-587, Sep. 2004.
[19]. R. Rojas-Cessa and C-B. Lin, "Captured-frame matching schemes for scalable input-queued packet switches," Computer Communications, May 2007.
[20]. H. Kim, J. Son, K. Kim, "A packet-based scheduling algorithm for high speed switches," Proc. IEEE TENCON, vol. 1, pp. 117-121, August 2001.
[21]. C. Hu, X. Chen, W. Li, and B. Liu, "Fixed-length switching vs. variablelength switching in input-queued IP switches," Proc. IEEE Workshop on IP Operations and Management, pp. 117-122, October 2004.