Design and Implementation of a Shared Memory Switch Fabric

Mina Ejlali
Dept. of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan, Iran
M.ejlalikamorin@ec.iut.ac.ir

Hossein Saidi
Dept. of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan, Iran
hsaidi@cc.iut.ac.ir

Mohammad Ali Montazeri
Dept. of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan, Iran
Montazeri@cc.iut.ac.ir

Ali Ghiasian
Dept. of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan, Iran
Ghiasian@cc.iut.ac.ir

Abstract—**Broadband networks satisfy the need to carry integrated traffic involving different types of information such as voice, video and data. Furthermore different services with multiple requirements need specific capabilities to be provided and guaranteed by broadband networks. The architecture of next generation networks has an important effect on changing the broadband networks and providing these capabilities. On the other hand, the architecture of broad band networks is highly affected by high speed switches. In fact, high speed switches are the target technology to achieve the required capabilities. The performance demands and changes in IC and VLSI technology have led to emergence of different types of switch architectures. This paper, proposes the architectural details of a scalable, high speed shared memory switch fabric which is operating at 20Gbps. The presented architecture is implemented using FPGA technology based on Xilinx's Virtex 4 family. The design presents a scalable architecture by implementing dynamic address allocation and efficient internal timing management. In our proposed architecture, we also consider some of VLSI design issues to make it more appropriate for further chip designs.**

Keywords—shared memory; interleaved memory banks; link-list; FPGA

I. Introduction

The current increase in network traffic makes great demands for switch architectures to operate in higher speeds. Network providers require providing guaranteed quality of services (QoS) to their customers. Looking at some of QoS parameters such as delay, jitter, loss and throughput one can observe the impact of network switches and routers on supporting QoS requirements. Therefore, high speed switches and routers with novel capabilities to differentiate traffic and provide different services based on their QoS requirements are of much interest due to customer demands. Then, the word 'QoS' is inseparable from network switches and routers performance.

This work has been partially supported by Iran Telecommunication Research Center.

The architecture of a switch or router can be partitioned into two main components, a) the line cards, and b) the switch fabric. The line cards are responsible for protocol processing, traffic management, policing and traffic shaping while the switch fabric is responsible for doing the action of high speed switching among the ports. The capacity of a switch fabric is highly depends on how good its architecture is designed and then how good it is implemented in practice. In this paper, we focus on the latter case, the switch fabric design and implementation. We mean 'switch fabric' whenever we use the term of 'switch' as an object in the rest of the paper.

A lot of research has been done to explore various switch design alternatives. Each of them has its own advantages and drawbacks, in terms of performance, latency, scalability and buffering. Always, there is a tradeoff between some parameters such as QoS support and complexities of various designs based on the current state of the technologies as well as economical issues. The function of switches is to make routing decisions and forward the packets from input to the appropriate output. It is possible that multiple arriving packets from different input ports have to be routed to the same output port. To deal with the output contention, switch systems use different types of architectures.

Switching systems in multi-plane and MIN[1] architectures are based on making different stages which are connected through lots of wires. Not only these connections lead to some delay in the entire communication system but also they are more expensive than implementing memories as buffers, from hardware implementation points of view [1].

On the other hand, deploying buffers has its own design issues to satisfy performance of switches. Designers of packet buffers must be careful about the latency and response time and

[1] Multistage Interconnection Network

should also consider buffer sizing issues to avoid packet loss problems.

Packet buffers are usually classified based on their location in switch fabric: input buffering, output buffering, shared memory and combined input and output buffer architectures. Input buffer architecture has the drawback of HOL blocking which limits the throughput of switch to 58.6% [2]. Output buffering has simple control and optimal performance [2], but it needs the speed up N and there is a large buffer requirement to each output port which is not so cost-effective. Also output buffering needs complicated scheduling algorithms to provide required QoS guarantees [3]. On the other hand, shared buffer architecture has high buffer utilization because of sharing its memory between all ports. It also allows high throughput and low cell loss probabilities. However, most switch architects prefer input buffering due to its ease of implementation.

Shared memory architecture has also its own challenges. A shared memory switch needs to operate much faster than its input data rate, because input ports are time multiplexed to access the shared buffer. Switching and buffering operations should be done by controlling memory read/write functions [4]. However, reducing access time to the shared memory is physically restricted [2]. In addition, the main area of the chip is occupied by shared memory and its related circuits. So, shared memory switch architecture would be expensive, from hardware implementation point of view. Moreover, if the packet size is not a multiple of memory width, it would make difficulties to design a shared memory [1]. Dividing packets to the fixed size cells can be helpful in dealing with this problem in high speed networks.

This paper presents the architectural design of a 4*4 shared memory switch fabric which operates at 20Gbps and also reviews its overall hardware implementation results. The architecture uses barrel shifter mechanism to perform parallel writes into different banks of buffers without the need to have multiplexer with higher speed. This mechanism allows scaling to larger switch sizes. On the other hand, proposed switch architecture presents dynamic solution as an address management algorithm to increase throughput and reduce hardware implementation cost.

The paper organization is as follow. In Section II, we review some related works from literatures. Section III presents overview of shared memory architecture. In Section IV, we describe the main proposed architecture. The synthesis results are given in section V. Finally, the paper concludes in section VI.

II. RELATED WORK

Current advances in FPGA technology makes FPGA a powerful device to implement high speed switches. FPGA technology can provide sophisticated hardware properties in design and implementation of switch fabric. Therefore, many of switch fabric architectures have been proposed using FPGA technology to be implemented during recent years.

Kabacinski and Michalski [5] proposed a hardware implementation of a new control algorithm for multiplane switch fabric. The hardware implantation presented in [5] is based on Viretx5 device in FPGA technology and also simulated in VHDL language. The control algorithm is appropriate for $\log_2$ (N, 0, P) switching network, where N is the number of inputs/outputs of switches and P is the number of planes. Many switch fabrics are using multiple chips in parallel. Input data is distributed among different planes and output data is gathered from these planes. Constructing stack of multiple networks with the structures like banyan, baseline, etc is one of the solutions to overcome blocking problem [5]. The control unit is capable of selecting a plane to setup a new connection in every clock cycle. The control algorithm is implemented using baseline multi-$\log_2 16$ switching fabric and can operate at 100 MHz. The designer also claimed that the clock could be speed up to 135 MHz. Kabacinski and Michalski also implemented a new switch fabric controller for rearrangeable Log_2 (N,0,p) fabrics with an even number of stages. The hardware implementation presented in [6] is based on Virtex 5 FPGA chip and it operates at 50 MHz. One of the drawbacks of multiplane switch fabrics is that in such architectures multiple switch fabrics require a lot of chips to setup switch interconnections and also they need to perform serial to parallel operation and vice versa [7]. Therefore a lot of power and board area are consumed.

A high performance switching based on buffered crossbar fabrics is proposed in [8]. This paper describes a set of scheduling algorithm which is used in VOQ/IBF[1] switch architecture. The name of proposed algorithm is Current Arrival First-Priority Removal[2] that is based on LIFO structure. The main difference of this algorithm and LIFO is that it consists of some stateless information exchange which leads in higher performance of input queue switch fabric under uniform and nonuniform traffic load. In addition, internal processing is designed for fixed size packets and variable length packets are fragmented to fixed sized cells. This scheduling algorithm is implemented for a 32*32 buffered crossbar switch based on Xilinx Virtex II Pro XC2VP20 FPGA. The results showed that the algorithm can support 10Gbps line speed. However switch fabric includes N^2 buffered cross points and is expensive to implement.

Turner in [9] presented a new control algorithm for crossbar switch fabrics. Proposed crossbar controller is synthesized on a Xilinx Vertex 5 FPGA. The controller is designed base on work-conserving scheduler such as LOOFA[3]. In order to achieve greater performance in crossbar switch fabric, scheduling algorithm allowed partial sorting. According to synthesized results, maximum placement and routing delay for 16*16 crossbar switch fabric is 5.7 ns that is fast enough for 10Gbps line rate. Maximum frequency of proposed design is about 175.43 MHz for 16*16 crossbar switch fabric.

An FPGA based shared memory buffer implementation was presented in [10]. The proposed architecture consists of five RLDRAM II memory chips, one of them is a 16MB device that is used to store linked list of packet addresses. Packets are stored into the linked list using 128B cells. Whenever cells are written into the packet buffer, the same addresses are used in control memory to provide an available next free cell address.

[1] Virtual output queuing/ Internal Buffering Crossbar
[2] CAF-PRMV
[3] Least Occupied Output First

At the time of reading a cell from shared buffer, the address of next cell will be accessed on the next TDM cycle. The shared memory controller is implemented by Altera Stratix II speed grade-4 device and is operating at 12.8Gbps line rate.

In [11] a 4*4 scalable shared memory ATM switch has been described which is implemented by FPGA technology. In the proposed architecture, there is an address controller that consists of free cell address queue, write cell address controller and read cell address controller. When a new cell with the fixed size of 16B is entered, its write address in the shared buffer is linked to the end of a special linked list. When a cell is being read from shared memory its address is set free and inserted in free address queue. The design has used a FIFO to store empty addresses. Shared memory utilizes scalable pipeline RAM system and can support high bandwidth for high switching throughput of the shared memory. The implemented design is operating in 40MHz. The proposed design in [10] has initial latency which is driven by SPRAM memory and also it is not cost effective from hardware implementation point of view due to the use of another memory as an empty address queue.

Due to the differences highlighted above, it was decided to investigate a new 4*4 shared memory switch architecture which is implemented in FPGA based on Xilinx's Virtex 4 family and operates at 173.575 MHz. The throughput of our developed switch fabric is greater than that of [11] whereas the maximum frequency presented in [11] is 40 MHz.

III. Overview of the Shared Memory Architecture

Shared memory switch architectures provide access to shared memory for all input ports at the same time. A central controller is managing the read/write operations in/out of shared memory. Although the central controller can be a bottleneck in heavy loads, but based on management algorithm that is deployed in central controller, shared memory architecture could achieve 100% throughput.

The shared memory should operate N times faster than input port rate to avoid packet loss where N is the number of port. Another critical issue in design of a switch is scalability of shared memory architecture. If R represents line rate, then difficulties of scaling shared memory architecture are because of: (1) Memory bandwidth should be increased as R increases (2) memory has to be accessed twice in every cell time. (3) As line rate increases, memory size must be increased too [3].

Recent example of shared memory switches is Cisco Catalyst 1200 series which supports 4 MB of shared packet DRAM [12]. In addition, Cisco Catalyst 4000 and 4500 series use shared buffer architecture. The Cisco catalyst 4000 utilizes 8 MB of SRAM and all frames are routed using a central processor and stored in a shared buffer until they are being switched. Moreover the Catalyst 4500 Supervisor IV uses 16 MB of SRAM for its packet buffer [12]. The Alcatel 7210 uses the same reliable, ASIC-based parallel access shared memory architecture as Alcatel 7652 and 7622 [13].

Multi port memories are good candidates for deploying shared memory switch architecture, but using multi port memories is not a cost effective solution [1]. Instead, using interleaved memory banks is an alternative to multi port memories. For instance, the required buffers in a knockout switch are constructed using memory banks [1]. Most of accesses to the interleaved memory banks are sequential. In other words, a unique address is used in order to access all banks. In this paper we develop a novel architecture by merging the remarkable properties of multi port memories and interleaved memories. To overcome the cost issue of multi port memories, we use available low price FPGAs and their internal block RAMs as the main component for constructing dual port memories

IV. Proposed Shared Memory Switch Architecture

This paper describes a new 4*4 shared memory architecture based on Xilinx's FPGA Virtex 4 technology. The shared memory architecture is constructed by interleaved memory banks that are implemented internally with SRAM memory blocks. The proposed switch architecture is shown in Fig 1 and has six main components:

- Input part;
- Crossbar;
- First-last module;
- Memory controller;
- Memory of addresses;
- Memory banks.

A. Input part

Input part of the switch architecture includes four input ports which can accept serial data. Each cell is 64byte. First of all, incoming packets are stored in input FIFOs. In this paper we assume that packets have fixed size called cells. In variable packet size networks (such as TCP/IP network), the packets can be segmented to fixed size cells. It is well known that by segmenting variable sized packets to fixed length cells, the performance of the switch fabric would be improved and many design issues are avoided. The cell size is 64 Byte which consists of 16 words of 4 bytes each.

On the other hand, to achieve 100% throughput in the shared memory architecture, the total rate of switching and forwarding should not be less than the arrival rate. Then, switch fabric has to perform three functions at each cell time: (1) make routing decision for each input cells about its desired output port (2) write cells into the memory (3) forward cells to output ports. To fulfill the above mentioned functions, each cell time is divided into 16 clock pulses and at each clock pulse one word of each cell is transferred to internal modules or memories. This method enables us to leverage pipelining technique to achieve high throughput. Although the pipelining technique has the benefit of achieving higher utilization, it requires precise timing. Since, at each time slot different access to memories are required to write new incoming cells and read the switched outgoing cells. This means that at each timeslot it should be determined that which module has access to which memory and for what purpose.

After segmenting cells into fixed size fractions input data follow special algorithm to be written in memory banks that will be described later. Then, they go through a sorter module

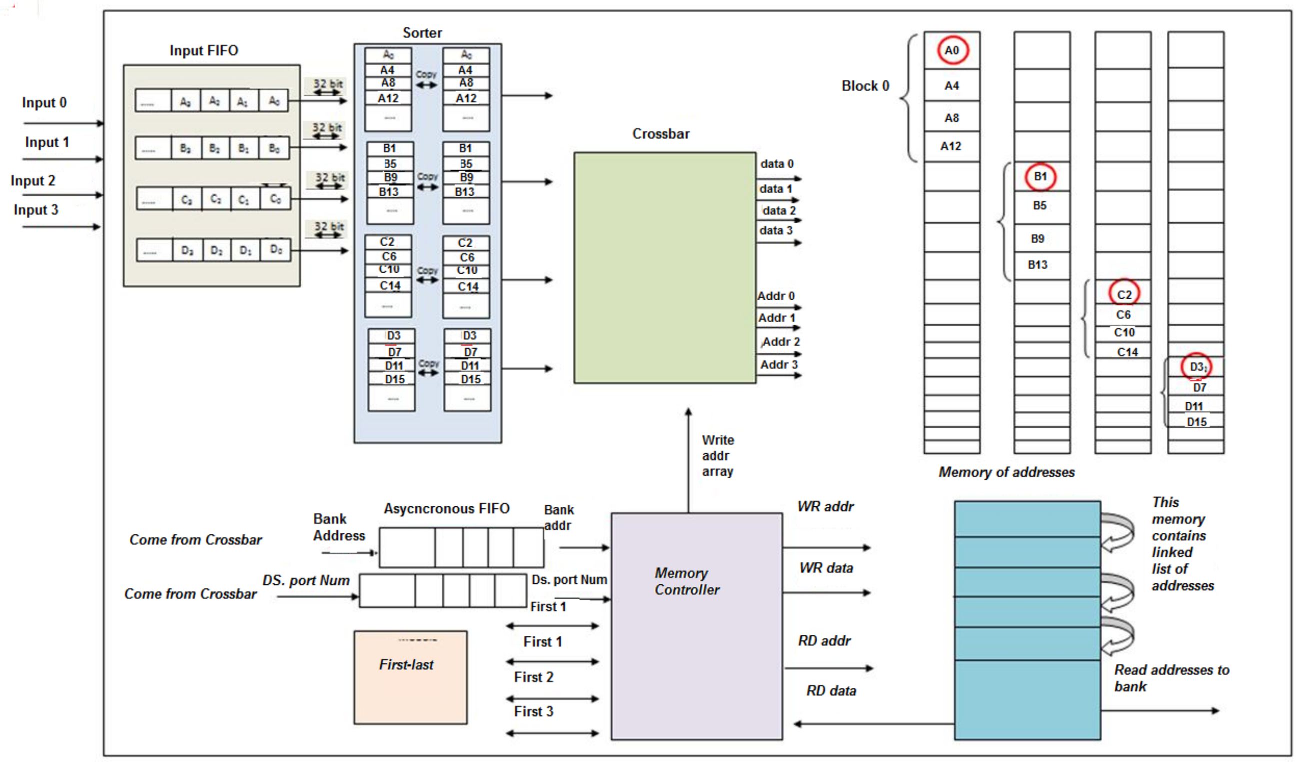

Figure 1. Proposed shared memory switch architecture

including two register parts. In this module the order of writing in memory banks is determined.

B. Crossbar

Packet switching function is performed in the crossbar module. The crossbar module makes decision about the memory location in which the incoming cells should be written. The algorithm for getting free address location of memory banks is described shortly. Crossbar module is in communication with other three main modules: input part, memory banks, memory controller module. Crossbar receives input cells within 16 clocks. After that, based on a specific algorithm that is implemented in crossbar, writing would be performed diagonally in memory banks. In fact, writing algorithm follows the barrel shifter model [14].

Let's denote the words of input cell from the first input port as $\{A_0, A_1, ..A_n\}$, the 2nd input port as $\{B_0, B_1, ...B_n\}$, the 3rd input port as $\{C_0, C_1, ...C_n\}$ and the 4th input port as $\{D_0, D_1, ...D_n\}$ as shown in Fig1. The function of sorter module is such that in the first clock, A_0, B_1, C_2 and D_3 are entered the crossbar and based on barrel shifting algorithm in crossbar they go diagonally across the memory banks. Therefore, four free addresses must be available to crossbar in this clock. Memory controller module is responsible for providing these free addresses. Then, in the second clock, A_4, B_5, C_6 and D_7 enter the crossbar. In this clock, all addresses will be increment by one and diagonally writing will be continued without requiring any new address from Memory controller module. Write operation in four consecutive clocks is shown in Fig.2. The offset pointers are determined by memory controller module.

In the proposed switch architecture, internal bus has 4 byte data width. Keeping constant the widths of data buses throughout the chip makes the design of the chip more convenient and cost effective. Using barrel shifter method in writing into memory, removes the need to use a multiplexer to access the memory. If in the write process we don't use this approach and our design was such that four consecutive words of a cell (say A_0, A_1, A_2, A_3) were supposed to be written in to four memory banks at the same clock pulse, then an extra multiplexor was required between Crossbar module and memory banks to convert 4 them to a 16 byte data bus to be written into the memory at one clock pulse. This increment in the data widths would have some drawbacks, considering FPGA implementation, larger data buses requires more routing resources. Since the internal routing resources such as routing matrixes and routing switches are limited in a typical FPGA device, increasing the width of data bus could result in using more resource for routing and in a possible case, lack of enough resources to full fill routing process. On the other hand, considering VLSI chip design issues, the larger data buses leads to the thinner width of internal metal tracks and its impacts on increasing the adjacent wire capacitance. Also large capacitance would result in area overhead [15].

Also, increasing data width results in more wire length which in turn results in more capacitance effect in hardware design because the wire capacitance is a function of wire-length and coupling capacitance between adjacent wires. In fact, there are some tradeoffs in choosing appropriate internal data width. For a given system throughput, smaller data width forces us to increase the frequency of clock pulse to keep up with the required throughput. Moreover, smaller data widths leads to less delay due to less routing resources in FPGA, less capacitance effect and therefore less area overhead [15]. Also it results to less power consumption, since length and width of wires in switch fabric interconnection is a key parameter in switch fabric power consumption [16].

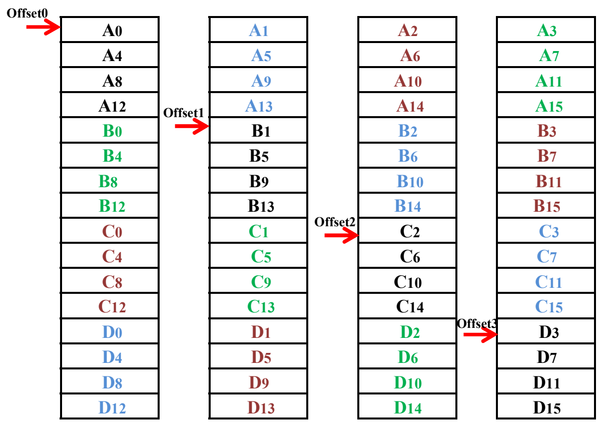
Offset0
Offset1
Offset2
Offset3
A0 A4 A8 A12 B0 B4 B8 B12 C0 C4 C8 C12 D0 D4 D8 D12
A1 A5 A9 A13 B1 B5 B9 B13 C1 C5 C9 C13 D1 D5 D9 D13
A2 A6 A10 A14 B2 B6 B10 B14 C2 C6 C10 C14 D2 D6 D10 D14
A3 A7 A11 A15 B3 B7 B11 B15 C3 C7 C11 C15 D3 D7 D11 D15

Figure 2. Barrel shifting model in memory banks

Considering the achieved clock frequency in our FPGA implementation which is 173.575 MHz and our target throughput which is 20Gbps, we propose the architecture based on 4byte internal data width.

C. *Memory controller*

Memory controller is responsible for managing the allocation of addresses in memory banks and it also manages read and write operations. This module is linked to three main modules in switch fabric: crossbar, memory of addresses, first-last module.

An algorithm is implemented to allocate addresses for incoming cells. This algorithm is also used to select a word when a cell must be read into the special output port. The following is the list of memory controller tasks in order to manage accesses to memory banks.

- Preparation and management of linked lists based on input requests for each output port;
- Preparation and management of linked list of free addresses in memory banks;
- Management of read/write operation from/to memory banks and updating the linked lists.

Free addresses should be available before writing a cell in memory banks. The Memory controller module handles the linked list of free addresses. Therefore, it will access the free addresses linked list to fetch four free addresses for crossbar unit to write four cells into the memory banks in a cell time.

While crossbar writes input cells in these fetched addresses, memory controller module insert them to the appropriate linked list based on their destination port. Memory controller use a SRAM memory to register addresses and is responsible to maintain linked lists.

Due to physical limitation in accessing to the memory of addresses, memory controller can add an address to a linked list and fetch a free address from free addresses linked list in one clock. There is a linked list for each output port. To insert a request in SRAM memory, memory controller requires two accesses to this memory. In first access, it will register the cell address which is the address of the location that the cell is written. Another access is making link to the previous request according to its output port number. Inserting an address in linked list is shown in Fig 3.

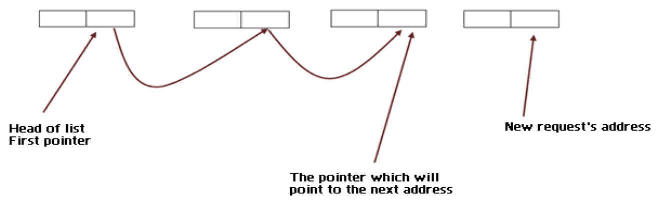

Figure 3. Inserting a new address to an output port linked list.

There is a pointer as a head pointer for each linked list. The Management and updating process of these pointers would be performed by First-Last module. When the memory controller wants to fetch an address from an inked list, it refers to its head pointer and then read the address that it points to. After that, memory controller sends this address to the memory banks for reading data which belong to this address.

Moreover, after reading out from memory banks, these addresses are set free from their linked list and should be inserted to the free addresses linked list. Consequently, there are two accesses in a readout operation. The first access is for reading an address from a linked list and the other access is for inserting the freed address to the free addresses linked list which imposes a write operation in SRAM memory. Also after reading an address from a linked list, memory controller should update the head pointer and forwards this new head pointer address to the first-last module.

D. *First-last module*

As described in previous section, for each linked list, there is a head pointer which is managed by first-last module. When memory controller inserts a new address to an empty list, this address is also registered in First-last module as a head pointer. First-last module contains four registers which stores four head pointers, one per each output port.

These pointers will be updated after reading out from memory banks. After fetching an address from a linked list, the Memory controller module updates the head pointer of that list to the next address in this list and then sends a new head pointer to the first-last module to be registered.

Also, another pointer which will be stored for each linked list is the last pointer. Last pointer points to the last address in a linked list. This pointer is used to add a new coming cell to the tail of the queue corresponding to that port.

E. *Memory of addresses*

Memory of addresses is a dual port SRAM memory module. As a result, reading and writing from/to this memory will be performed simultaneously. During the initialization process of the system, all addresses in this memory are linked to each other to create the free addresses linked list. The last address of this memory is known as the last free address and if memory controller wants to insert a new free address to this list, the new address will be linked to the last address. The size of this memory is a quarter of the size of memory banks, since it stores only the addresses of the beginning of a block (4*4byte) of memory.

Considering the architecture proposed in this paper, it should be noted that for scaling up the size of proposed switch fabric to include more input ports without changing memory management algorithm and keeping constant the timing configuration of control signals, memory bandwidth should be increased.

However our design is more scalable than the design presented in [11] which utilized static queues as a solution for address allocation in shared memory address management. By creating linked list for each output port's requests, address allocation will be done dynamically. Also address space will be shared among all linked list that involve requests. The structure of two sample linked list in the memory is shown in Fig 4.

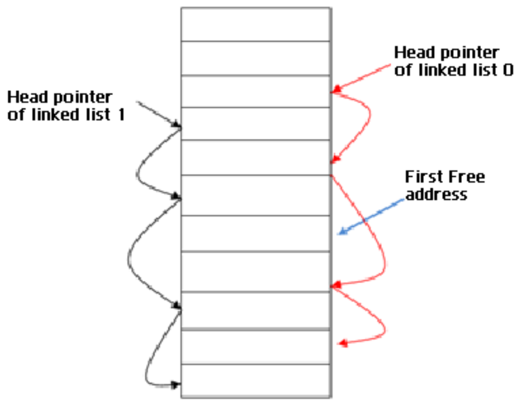

Figure 4. Structure of linked lists in the memory of addresses

F. *Memory banks*

Memory banks are in communication with two main modules; crossbar and memory controller. Write addresses of these banks are initialized by crossbar unit. Memory banks are also dual port SRAM memories which can handle read and write operation simultaneously. The memory consists of some blocks. In fact, memory controller provides the addresses of the beginning of each block and then crossbar will add two bit at the end of each address and generate four addresses within a block that will be used for four words of a cell. The first block in memory banks is shown in Fig 5.

Although the memory banks are physically separated, address space is dynamically shared among all inputs. Therefore, they have the same opportunity to access the shared memory. In fact, instead of having a static address space for each input port, the proposed switch fabric shares the total memory space among all the ports.

In comparison with [10] which has five RLDRAM II memory chips, our design is more cost-effective by utilizing interleaved SRAM Memories. In addition, by implementing interleaved bank accesses to the shared memory module all input ports have equal chance to access to the memory. This is an advanced technique in which leads to performance improvement. In this way, memory bandwidth could be increased by simultaneous access to multiple banks.

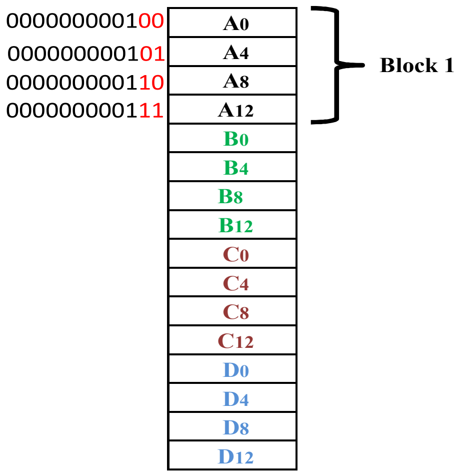

Figure 5. An example of the first block and its addresses

V. IMPLEMENTATOIN AND RESULTS

The proposed shared memory architecture has been synthesized and implemented by Xilinx ISE 12.3 tools. Also the architecture is implemented using VHDL programming language. The target device specifications are shown in Table I.

TABLE I. SPECIFICATION OF FPGA BOARD

Family	*Device*	*Package*	*Speed*
virtex4	XC4VLX100	11ff1148	-11

The two critical factors for evaluating a hardware design are hardware speedup which is shown by operational frequency and the design hardware complexity. The results of device utilization are reported in tables II respectively. As shown in table II, the overall usage of the device presented in different rows, is less than 30%. The significance of the results reported in table II is that the utilization of resources is minimized. The design has utilized less than 30% of logic units, the remaining area can be used for designing other functions such as packet classification and etc. These results are extremely affected by the switch fabric architecture proposed in this paper and the way it utilized linked list design instead of using static queuing of the addresses.

Synthesized results verify the capability of implementation of the presented switch fabric design. In fact, the proposed architecture has been achieved a proper tradeoff between switch fabric throughput and hardware complexity.

TABLE II. DEVICE UTILIZATION SUMMERY

Logic utilization	*Used*	*Available*	*Utilization*
Number of slice	5985	49152	12%
Number of slice flip flop	5947	98304	6%
Number of 4 input LUTs	10871	98304	11%
Number of FIFOs16/RAMBs 16s	69	240	28%

As synthesis results shown in table III, by designing a proper internal timing as described in previous section, switch fabric achieved maximum frequency up to 173.575 MHz with a maximum clock period of 5.761ns. Use of 4-byte internal data width and barrel shifter mechanism is one of the main drivers of the improvement in delay routing resources in FPGA. It also leads to less area overhead as seen in table II.

The post placement and routing implementation results indicates that maximum routing delay in switch fabric is 5.761ns. Since a cell time for writing a 64Byte packet is 16 clock pulses, therefore for a single input port, the maximum throughput is 5.55Gbps. As a result, the total capacity of 4*4 switch fabric would be 4×5.55Gbps=22.21Gbps. This results show that proposed switch using FPGA technology can operate at 20Gbps line rate.

TABLE III. SYNTHESISE REPORT

Maximum Frequency	173.575MHz
Maximum period	5.761ns
Output offset	4.221ns

The summarized results in table IV indicates a superior operating frequency for the proposed switch fabric architecture in comparison with architecture presented in [10] and [17].

TABLE IV. COMPARISON OF SHARED MEMORY SWTICH IMPLEMENTATION RESULT IN TERMS OF MAXIMUM FREQUENCY AND SWITCH CAPACITY

Shared memory switch Architecture	***Design proposed in [10]***	***Design proposed in [17]***	***Design proposed in this paper***
Maximum Frequency	100MHz	80MHz	173.575MHz
Switch Capacity	12.8Gbps	2.5Gbps	20Gbps

VI. CONCLUSION

This paper has presented the architectural details of a scalable, high speed 4*4 shared memory switch fabric for high speed networks. This design has used FPGA technology to achieve the desired throughput. The implementation results showed that the switch fabric can operate at 20Gbps.

The proposed switch architecture implements shared memory architecture by using interleaved memory banks and barrel shifter method. A memory controller is designed to manage accesses to the shared memory by applying linked list structure to maintain packet addresses in the shared buffer. We also consider some VLSI design issues in our proposed architecture in order to make it suitable from chip design point of view. Our proposed design made some tradeoffs to choose optimal internal data width in order to provide the desired throughput and less delay due to reduction of the routing resources capacitance effect, less area overhead and less power consumption.

The presented architecture is implemented based on Xilinx's Virtex 4 family. It is expected that by improving FPGA technology to implement presented switch fabric, further speed gains can be achieved.

ACKNOWLEDGMENT

This work has been partially supported by the Iran Telecommunication Research Center.

REFERENCES

[1] M. Katevenis and P. Vatsolaki, A., Efthymiou ,"Pipelined Memory Shared Buffer for VLSI Switches," Proc. Of the SIGCOMM conference on Application technologies, architecture, and protocols for computer communication , 1995.

[2] M. Karol, M. Hluchyj and S. Morgan, "Input versus Output Queuing on Space-Division Packets Switch," IEEE Trans on Communications, 1987.

[3] D. Divakaran, S. Soudan,P. Primet,E. Altman, "A survey on core switch designs and algorithms," INRIA - *http://www.inria.fr* , inria-00388943, version 1 - 28 May 2009.

[4] M. Arpaci, J.A. Copeland, "Buffer Management for Shared Memory ATM Switches," IEEE Trans on Communications, 2000.

[5] W. Kabacinski, and M. Michalski, "The FPGA Implementation of the Log(N,0,P) Switching Fabric Control Algorithm," Proc of 11thInternational Conference on High Performance Switching and Routing., HPSR.2010.

[6] W. Kabacinski, and M. Michalski, "FPGA Controller for Rearrengeable Log_2(N,0,P) Fabrics With an Even Number of Stages" Proc of 12th International Conference on High Performance Switching and Routing., HPSR.2011.

[7] H.R. Jang, and H.S., Kim,"A Self-routing Switch Fabric Architecture on a Chip," Proc of Global Telecommunication Conference, IEEE GLOBECOM, 2008.

[8] L. Mhandi, M. Hamdi, C. Kahris, S. Wong and S. Vassiliadis,"High-Performance Switching Based on Buffered Crossbar Fabric" Journal of International Computer & Telecommunication Networking, 2006, Vol. 50, Issue. 13, PP. 2271-2285.

[9] J. Turner, Design and Evaluation of a Practical High Performance Crossbar Scheduler, Tech. Rep, Washington University in St.Louis, 2009.

[10] D. Burns, C. Toal,K. McLaughlin, S. Sezar,M. Hutton, and K. Cackovic, , "An FPGA Based Memory Efficient Shared Buffer Implementation," Proc. Of International Conference on High Performance Switching and Routing , 2007.

[11] W.J Shim, J.G. Jeong, K.M Lee,"FPGA Implementation of a Scalable Shared Buffer ATM Switch" Proc. Of 1th IEEE International ICATM-98 Conference , 1995.

[12] D. Barnes, B. Sakandar, Cisco LAN switching fundamentals, Cisco Press. USA, pp. 42–43, 2005.

[13] http://www.alcatel-lucent.com/wps/portal/products.

[14] http://www.realworldtech.com/page.cfm?ArticleID=RWT081502231107&p=5

[15] N. Muralimanohar, Wire Aware Catch Architecture, A dissertation submitted to the faculty of the University of Utah., December 2009.

[16] T. Ye, L.Benini,D. Micheli,"Analysis of Power Consumption on Switch Fabrics in Network Routers," Proc of 39th Design Automation Conference , 2002.

[17] G.J. Jeong, M.K. Lee ,"Design of a Shared Scalable Buffer ATM Switch Embedded Low-Power SPRAM, " Journal of Electrical Engineering & Information Science , 1998, Vol. 3, No. 6, PP. 752-762.