Implementation Of Interleaving Switch Based Architecture For System On Chip

[#1] M. Praveen Kumar [#2] G. Akhila [#3] B.Dharani [#4] V.Joel Frank

[#1] *Assistant Professor, Department of Electronics and Communication Engineering, Vignana Bharathi Institute of Technology, Aushapur, Ghatkesar, RangaReddy, (A.P.), India.*

[#2#3#4] *UG Students(ECE), Department of Electronics and Communication Engineering, Vignana Bharathi Institute of Technology, Aushapur, Ghatkesar, RangaReddy, (A.P.), India.*

ABSTRACT:

A **System on a chip** (**SoC**) is an integrated circuit (**IC**) that integrates all components of a computer or other electronic system into a single chip. The communication mechanisms employed in these systems on a single chip (SoC) are an important contribution to their overall performance. Till now, bus-based paradigm is applied in many areas of real-time applications of SoCs realizing on FPGA due to its flexibility and simplification in designing tool. Although offering the module-increasing flexibility, its bandwidth and scalability still become problem. To alleviate these problems, switch-based paradigm has been introduced to improve the above three important factors. The main advantage of this switch-based paradigm is that it can perform multicasting (many-to-one) and gathering (many-to-many) operations also. But this paradigm also bears a disadvantage called output latency. That is the reason we have elected interleaving mode in this project. The architecture selected is a **6x6 Interleaving Crossbar Switch architecture**, which is implemented for the proposed project and verified on Virtex4V XC4VFX12. Verifying the performance with the 1024 sample words at 100MHz and we have used 23 Flip flops and 134 four-input Look Up Tables (LUTs). Flip flops and LUTs, both individually constitute 1% of the target FPGA. Similarly estimation frequency of 440 MHz with a time period of 2.47ns was obtained from the Xilinx ISE tool.

Keywords: SOC, Interleaving, Unicast, Multicasting, Crossbar switch.

I. INTRODUCTION:

Today field programmable gate-arrays (FPGAs) are used for a wide sector of applications. Field Programmable Gate Arrays (FPGAs) are widely increasing their consumption in many applications. Its usage in former times was focused on rapid-prototyping system for integrating test systems. After the test-phase, often an ASIC approach substituted these systems for mass-production. Due to dramatic growth in circuit-design complexity regarding Moore’s Law, the ability of implementing complex architecture in a single chip always presents new challenges. One of the issues found by designers while implementing large SoCs is the communication among their components.

Buses are an increasingly inefficient way to communicate, since only one source can drive the bus at a time, thus limiting bandwidth.

NoCs are increasing in popularity because of their advantages: larger bandwidth, and lower power dissipation through shorter wire segments. Communications in large SoCs are so important that many designers have adopted the NoC approach. The challenges consist in offering the best connectivity and throughput with the simplest and cheapest architecture of methodology; whereas many topologies and architectures have been investigated.

The idea of designing the Reconfigurable Crossbar Switch for NoCs to gain a high data throughput and to be capable of adapting topologies on demand is our main aim. Their evaluation results showed that output latency, resource usage, and power consumption were better than a traditional crossbar switch. Nevertheless, they did not focus the problem while operating many-to-one and one-to-many data communication. Our proposed crossbar architecture is designed with two contributions: First, our crossbar has to be lightweight, requiring few FPGA resources, and thus suitable for both small and large resources with high bandwidth efficiency on FPGA-based systems. Second, the proposed crossbar architecture has to solve the output delay (output latency) problem while multicasting on all source situations. Moreover, in order to obtain the work efficiency, 6x6 interleaving

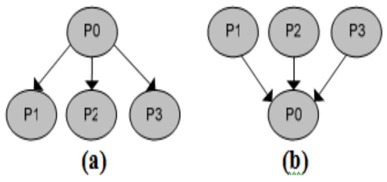


Fig. 2. a) One-to-many b) Many-to-one

switch-based crossbar is realized by VHDL, and verified on the Xilinx FPGA Virtex4 XC4VFX12.

II. COMMUNICATION ISSUES

In this project we are using interleaving mode of communication. First of all, we are well renowned with the types of communications like one to one communications and many to one communications. The main issues which we face with these are discussed below in detail.

In Fig.1, all communication in a process group becomes many communications involving an arbitrary subset of processor from the system's point of view.

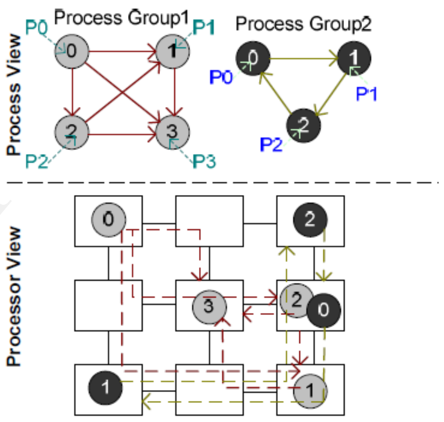


Fig.1 System's point of view

In order to efficiently support data communication, a system has to support one-to-one (unicast), one-to-many (multicast), many-to-one (gathering) and many-to-many communication primitives in hardware. Actually, most interconnection can support unicast, but not gathering and multicasting.

In fig 2(a), we can clearly see a multicast communication with three destinations where P0 has to send same data to all the three processors: P1, P2, P3. Without the multicast functionality on interconnection, the system can use unicast functionality to send data to all destinations sequentially, but at this instance only blocking will occur. If P0 is executing sending state to P1 and P1 has not yet reached receiving state, P0 will be blocked. Meanwhile, P2 will execute receiving state and is blocked because P0 has yet not received sending state. So definitely, system resources are wasted due to unnecessary blocking. Fig.2(b) shows gathering data communication from three sources to one destination. Because of sending data from all sources, congestion is found at destination where it can be reduced by applying source priority. Supposing P1,P2 and P3 are the first, second and third priorities. While their data arrive at P0, the data from P1 will be the first forwarding; meanwhile, the rest will be buffered. Thus, the output latency of system will increase, and buffer will also require. In order to solve these problems, we apply interleaving technique to switch-based crossbar architecture on FPGA, and compare resource usage as well as estimated frequency.

III. SWITCH-BASED CROSSBAR ARCHITECTURE

The major components that make up the switch-based crossbar consist of *Input Port module*, *Switch module* and *Output Port module*. Depending on the kind of channels and switches, there are two alternatives for designing switch-based crossbar: unidirectional and bidirectional. In this paper, unidirectional switch-based crossbar is selected and simplified with 6x6 switch-based crossbar structure shown in Fig. 1 because of resource constraint. Obviously, transmitting data from a *Source Node* to a *Destination Node* require crossing the link between the *Source Node* and the *Input Port module*, and the link between the *Output Port module* and the *Destination Node* where the *Switch module* in data path will dynamically establish the link for the *Output Port module* according to switching protocol.

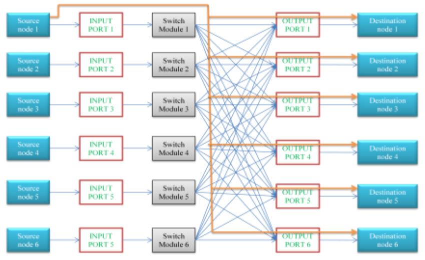


Fig.3 6x6 switch-based crossbar Architecture

According to the 6x6 switch-based crossbar architecture its switching protocol is shown in Fig.4.

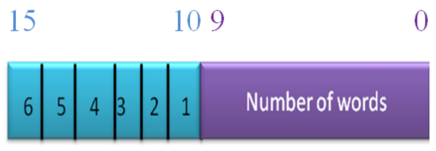


Fig.4 16-bit switching protocol

Because of resource constraint on the target FPGA [2], data width, the No of word register, the destination port register are 16, 10 and 6 bits respectively. Moreover, synchronous protocol is applied to synchronize all modules in the switch-based crossbar-*Clock signal*, *Ready signal* and *Acknowledge signal*. Fig. 2 shows the 10-bit Last Significant Bit (LSB) of switching protocol, which defines the number of packets required transferring from a *Source Node* to a *Destination Node*, where the maximum is 1,024 packets per time, and the rest define the *Destination Node*. For example when the *Source Node1* wants to transfer 100 packets to the *Destination Node* number *5* and *4*, the 16-bit switching protocol has to be 0101_0000_0011_0100 (0x5034H).

A. *Input Port module*

In this section, the architecture and behavior of the *Input port module* are detailed. As shown in Fig. 5(a), there are three components in this module comprised of a *16-bit tri-state buffer*, a *1-bit tri-state buffer* and a *finite state machine (FSM) component* where the *No of word register* and the *destination port register* are resided. Its behavior shows in Fig. 5(b) based on switching conceptual. At the beginning, the state is in an *idle state*, and then the *header ready signal* enables HIGH to inform a *Source Node* that ready to read the switching protocol.

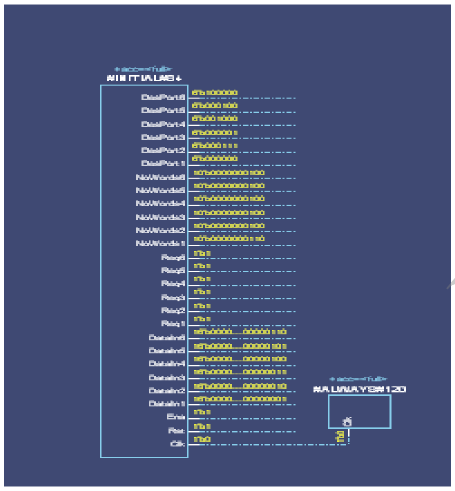


Fig.5 Data flow diagram of the module

As soon as the 16-bit switching protocol is asserted at the *Data in signal* and the *Data in valid signal*, the state goes to the next *check state*. In this state, the 16-bit switching protocol is separated and written on the *No of word register* and the *destination port register* resided in *FSM module*, where 10-bit low and 6-bit high are written on the *No of word register* and the *destination port register*; meanwhile, the *register port signal* is read to check, whether or not the required destination ports are free. If it is free, the state will go to the *req state*. In the *req state*, the *No of words register*, the *destination port register* and the *req signal* are read and presented on the *No of words signal* and the *No of word valid signal*. When the *rsp signal* enables HIGH, the channel for transferring data is guaranteed, and the state goes to the *Run state*. In the *Run state*, the *data in ready signal* enables HIGH. The *16-bit data in signal* and the *data in valid signal* are sequentially asserted into the channel. When the lasts data is completely forwarded, the *rsp signal* from the *Output port module* will enable LOW, and the state will start to the next cycle. Its behavior is shown in Fig. 6 based on switching conceptual.

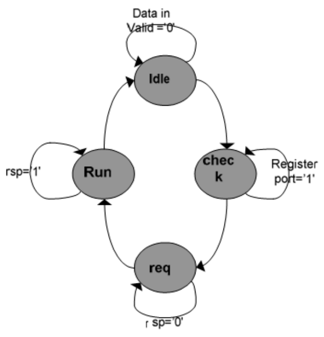


Fig. 6 State-machine

B. *Output Port module*

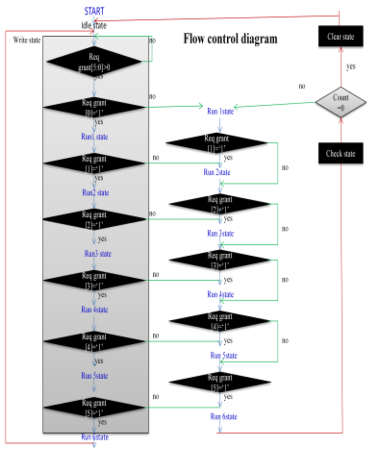


Fig.7 Flow control diagram

In this module, the *Output Port module* architecture and the flow diagram are explained in detail. There are three components of this module, comprised of a *16- bit six-port multiplexer* component which used to multiplex the *Data in signals* coming from the *Switch Module*, the *Round-Robin* component assigned to guarantee the fairness of all input requirements and the *FSM component* where the *10-bit counter register* and the *6-bit Port Status register* used to count down a number of through packets and to identify that can occupy the path of this *Output port module* are resided. Since interleaving mechanism bases on Time-Division-Multiplexer (TDM), the *16-bit six-port multiplexer component* can multiplex all *Data in signals* depended on Time Slot.

Fig. 7 explains its behavior. At first, the state is in *Idle state*, and the *mode signal*s are read where there are two kind of possible modes, normal mode and interleaving mode. In the normal mode, the *Round Robin component* will be enabled but will be disabled in the interleaving mode. Whenever a *Source Node* requires transferring its data to a *Destination Node*, the *req signal* of the *Input Port module* connected with the *Source Node* will enable HIGH. Then, at the *Output-Port* module connected with the *Destination Node*, each bit of the *6-bit grant register* related with each *req signal* of each *Input Port module* will set "1". The *No of words signal* and the *No of words valid signal* will be read and stored into the *temporal register*. Then, the state goes to the *Write state*. In the *Write state*, each bit of the *grant register* is read to check the *Source Node*'s requirement and to identify the location of the next state. In order to simplify, supposing the *grant register* contains 001000. It implies that the *input port modules* number 4 will forward their data to the *output port module*. Therefore, the state will start at the *Run3 state*, and the *Data in ready signal*s number of 4 will be enabled HIGH.

Until the *Run6 state* is executed, the state goes to the *Check state*.The *counter register* counting the forwarded packets is checked, whether or not the forwarded packets has been completely sent. Whenever, they have been sent properly this *counter register* will be "0000000000" and the *Clear state* will be done. At the *Clear state*, the *rep signal*s at 4 are enabled LOW and the *grant register* will be "000000", as well as the state will begin the next cycle.

For instance, Fig.8 shows us the timing diagram of two source nodes. The first source node transmits data to the destination node 4 and the second source node transmits data to all the destination nodes. In other words we can say the destination nodes are 001000 and 111111.

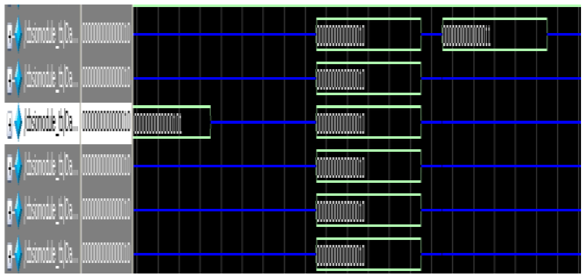


Fig.8 Timing cycles of 001000 and 111111

C. Switch Module

Switch module crosses the link between the *Input Port module* and the *Output Port module*. Underneath the *Switch module*, there are five components connecting with two sets of signal. First, the *control signal* consists of the *req signals* and the *resp signals*.

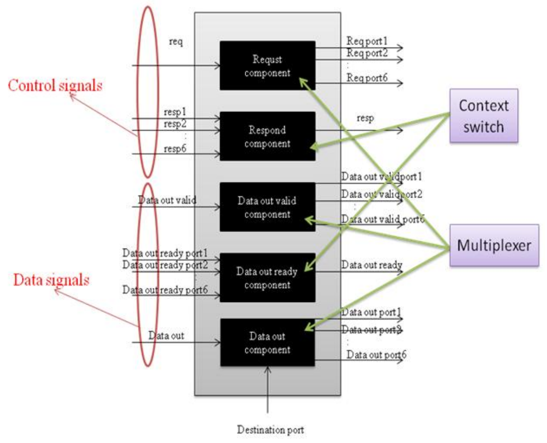


Fig.9 Switch module

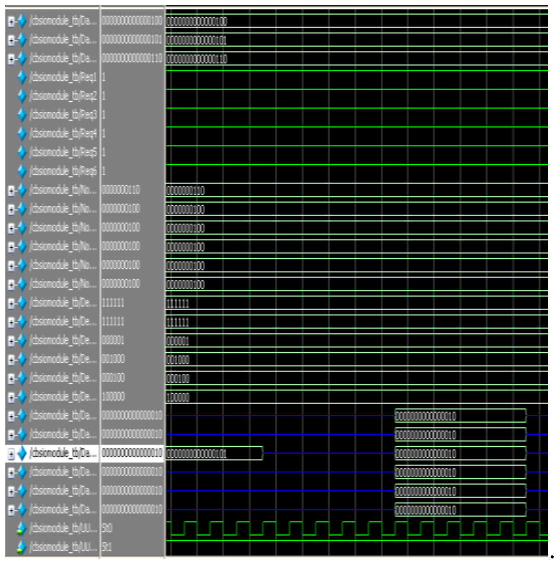
Second, the *data signal* composes of the *data out valid signal*, the *data out ready port signals* and the *data outsignal*. Two types of switch component are context switch and multiplexer. All components of each set can be mapped with two types of switch component as shown in Fig. 8, where they are controlled by the *3-bit Destination port signals* of *Input Port module*.

IV. IMPLEMENTATION

The problems discussed earlier and the primitive modules of the switch-based crossbar architecture are implemented and realized on the target FPGA in this section, ie they are implemented on Xilinx tool. All primitive modules are structured by VHDL, verified their behaviors with Model Sim 10.1d, synthesized their resource usages and estimated frequency on the Xilinx FPGA Virtex4 XC4VFX12 by ISE tool as shown in Table1.To realize the target FPGA, 6x6 interleaving switch-based crossbar is introduced.

V. RESULTS

The result of the 6x6 interleaving switch-based crossbar is designed and implemented, which is shown in Fig. 10. The *Source Nodes* and the *Destination Nodes* connecting with our crossbar can be reconfigurable, where several IP cores can take place.



```
/cbsimodule_tb/Da... 0000000000000100
/cbsimodule_tb/Da... 0000000000000101
/cbsimodule_tb/Da... 0000000000000110
/cbsimodule_tb/Req1 1
/cbsimodule_tb/Req2 1
/cbsimodule_tb/Req3 1
/cbsimodule_tb/Req4 1
/cbsimodule_tb/Req5 1
/cbsimodule_tb/Req6 1
/cbsimodule_tb/No... 0000000110
/cbsimodule_tb/No... 0000000100
/cbsimodule_tb/No... 0000000100
/cbsimodule_tb/No... 0000000100
/cbsimodule_tb/No... 0000000100
/cbsimodule_tb/No... 0000000100
/cbsimodule_tb/De... 111111
/cbsimodule_tb/De... 111111
/cbsimodule_tb/De... 000001
/cbsimodule_tb/De... 001000
/cbsimodule_tb/De... 000100
/cbsimodule_tb/De... 100000
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/Da... 0000000000000010
/cbsimodule_tb/UU... St0
/cbsimodule_tb/UU... St1
```

Fig.10 Result obtained

In the result shown above, we can see that in the first instance, only destination node 4 has received the data sent from source node and coming to the second instance, all the destination nodes have received the data sent from the source node. This is the main mechanism for which the project has been proposed and implemented and now we have achieved the desired output.

VI. CONCLUSION

This paper proposes and implements the interleaving mechanism to overcome the output delay (latency) while operating one-to-one and one-to-many data communication. The proposed crossbar performance has been verified with the 1024 sample words at 100MHz which achieved the bandwidth from 335.42 up 741.31 Mbit/sec measured when 256 packets (16 bit per packet) are fed at 100 MHz based on the test environment.

The resource usages realized on the Xilinx FPGA Virtex4 XC4VFX12 are 3.33 % and 16.05 % for slice Flip-flop and LUTs respectively, and estimated frequency respond is 113.854 MHz. Moreover, resource usage and bandwidth are acceptable when it is compared with the general propose switch-based system and bus-based system.

REFERENCES

[1] D. Bafumba-Lokilo, Z. Savaria, J. David , "Generic Crossbar Network on Chip for FPGA MPSoCs", Circuits and Systems and TAISA Conference, 2008, pp 269-272.

[2] Xilinx , "On-chip Peripheral Bus V2.0 with OPB arbiter", "Processor Local Bus(PLB)v4.6(v1.03a)", http://www.xilinx.com.

[3] H. C. d. Freitas, P. Navaux, "On the Design of Reconfigurable Crossbar Switch for Adaptable On-Chip Topologies in Programmable NoC Routers", ACM Special Interest Group on Design Automation, 2009, pp.129-132.

[4] H.Po-Tsang, H. Wei , "2-Level FIFO Architecture Design for Switch Fabrics in Network-on-Chip", Circuits and Systems, ISCAS 2006, Proceedings 2006, IEEE International Symposium on, pp.4863-4866.

[5] M. Hübner, L. Braum,D. Göhringer, J. Becker, "Run-Time Reconfigurable Adaptive Multilayer Network-On-Chip for FPGA-Based systems", IEEE International Symposium on In Parallel and DistributedProcessing, 2008, pp. 1-6.

[6] C. Hilton, B. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA based systems", IEE Proceeding Computing Vol. 153, 2006, pp. 181-188.

[7] C. Qiao, R.Melhem, "Reconfiguration with Time Division Multiplexed MIN's for Multiprocessor Communication", IEEE Transections Parallel and Distributed System, Vol. 5, 1994, pp. 337-352.