HIGH-SPEED VLSI ROUTER ARCHITECTURE WITH INTELLIGENT BUFFERING FOR NOC EFFICIENCY

[1]G. LAVANYA, [2]P. SURYA, [3]CH.ANITHA, [4]A. NAGA RAJITHA [5]P. SARANYA
[2]Associate Professor, ECE Dept, RISE Krishna Sai Prakasam Group of Institutions, Valluru
[1,3,4,5] Students, ECE Dept, RISE Krishna Sai Prakasam Group of Institutions, Valluru, AP
[1]lavanyagunji05@gmail.com,[2]suryame3020@gmail.com,[3]anitha52897@gmail.com,
[4]alapatirajitha8@gmail.com,[5]saranyasaruu043@gmail.com

ABSTRACT
Faster and more effective data transmission inside a system-on-a-chip (SoC) is becoming more and more crucial as chip technology develops. The main goal of this research is to design a high-speed VLSI router that uses adaptive buffering to improve data transfer in Network-on-Chip (NoC) environments. Each input port in the architecture has a FIFO buffer, which briefly stores incoming data to prevent packet loss and congestion. While a crossbar switch quickly directs packets to their destinations, a dynamic arbiter controls data access by giving priority to active ports. When combined, these elements provide high throughput, low latency, and seamless traffic flow. The Vivado Design Suite, a potent FPGA development tool, was used to simulate the design in order to confirm its functionality and performance. Through this environment, the router modules were tested under different traffic conditions, confirming the system's ability to operate efficiently at high speeds. The synthesis reports further showed that the design consumes minimal area while delivering robust performance.

Keywords: Network-on-Chip (NoC), Router Architecture**,** Arbiter**,** Crossbar Switch

I INTRODUCTION

Because it can manage the increasing number of cores in System-on-Chip (SOC) designs, Network-on-Chip (NOC) architecture has gained popularity in the field of communication infrastructures. This architecture was developed to address the shortcomings of the SOC paradigm and the previously fevered communication technologies, which had problems with low core utilization, poor reuse, high complexity, and poor scalability [3]. Network configuration and data flow characterize NOC, and it is crucial to comprehend its design ideas, including topology, control flow, switching mechanisms, routing strategies, and the components utilized in its implementation [4, 5]. These parts, which control data flow inside the NOC design, include the Arbiter, Router, Crossbar, and FIFO Buffer. This work focuses on analysing and simulating these parts, especially the routers, and looking at their internal structure to better understand how they operate and enhance their performance. To give a thorough grasp of the NOC design, the implementation of NOC in VHDL will also be covered.

II.LITERATURE SURVEY

In the work [6], a 4-phase bundled data protocol is used to interface a First-In-First-Out (FIFO) memory buffer with a Static Random-Access Memory (SRAM). The hardware description language VHDL is used to model the FIFO and SRAM, and asynchronous handshaking concepts are used for both components' communication. Xilinx ISE version 9.1i is used to analyse the design in terms of timing and power consumption. The findings demonstrate that asynchronous systems can be leveraged to develop fast, power-efficient, and clock-skew-resistant high-performance memory designs like the FIFO-SRAM block.

In the work [6], a 4-phase bundled data protocol is used to interface a First-In-First-Out (FIFO) memory buffer with a Static Random-Access Memory (SRAM). It is generally accepted that NoC will take the place of traditional bus-based design and meet the communication requirements of the next SoC. A router is the crucial component known as the NoC's communication backbone. This paper presents the NoC router architecture, which utilizes less space and has a low latency. The design is implemented using VHDL and simulated using Xilinx ISE Design Suite 13.1. In this study, a fixed priority arbiter- 12 based five-port NoC router is proposed. The router is synthesized and simulated using Xilinx ISE design suite 13. The simulation shows how the NoC router works [7].

Because of its simplicity and fairness in improving system efficiency, the Round Robin (RR) algorithm is widely utilized in many different industries. The usage of the RR algorithm in CPU scheduling and cloud computing is reviewed in this paper, along with methods for algorithm improvement. The choice of an optimal time quantum is one method that researchers have suggested for improving the RR algorithm. The Round Robin algorithm is a well-liked scheduling technique for enhancing system efficiency across a number of industries, including CPU scheduling and cloud computing. Two categories of studies have been established: RR based on static quantum and RR based on dynamic quantum. There are two additional categories for dynamic quantum studies: per-round and per-process.

To help researchers enhance or optimize the RR algorithm in different disciplines, a comparative analysis of the methods has been presented [8]. An architecture with blocks like arbiter, crossbar, and FIFO buffers was proposed in a study [9]. Round robin architecture was employed in the arbiter block. Initially, a 4x4 mesh architecture was intended for the router. Xilinx software was used to propose and implement the design. According to the outcome, the design took up 238 of the 17,334 available slices.

III. EXISTING SYSTEM

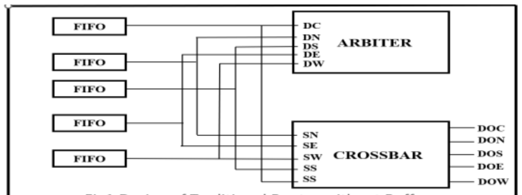


Fig1.Design of Traditional Router without Buffer

A thorough understanding of factors like topology, flow management, switching methods, and routing algorithms is necessary for the design of a NOC router. A network's topology describes how its nodes and channels are connected. Several topologies, including mesh, torus, star, spin, butterfly, and others, have been proposed. Mesh topology is used in this architecture due to its simplicity and ease of integration, while also serving addresses aligned with messages. Low latency, high throughput, low power consumption, low cost, and great performance are all desirable properties of an ideal architecture, but achieving them all is difficult.

The switching technique, which can be divided into circuit switching and packet switching, is the process used to link a router's input and output.

Whereas packet switching transmits data as soon as it becomes available, regardless of the path, circuit switching only transmits data packets once the path has been established. A routing strategy is a method for getting data packets to their destination while avoiding starvation, deadlock, or live lock [10]. Data packet waiting indicates a bad flow control technique, while deadlock avoidance indicates a good one. Flow control controls the distribution of resources to data packets. By looking at a data pac's journey

FIFO, Arbiter, and Cross-bar are included in the first design that was taken into consideration. Fig. 1 depicts the circuit. To temporarily store incoming data and avoid congestion during routing delays, the design starts with several FIFO buffers, each assigned to a directional input port such as Center, North, South, East, and West.The arbiter uses these FIFO buffers to make real-time routing decisions by monitoring their occupancy status and sending signals indicating whether they contain data (non-empty).

After receiving these active signals, the Arbiter module uses a rotating mask mechanism to decide which input should be allowed access to the router's crossbar switch in order to maintain fairness and avoid port starvation. Depending on which port has prevailed in arbitration, the arbiter generates a grant signal that enables one directional input path at a time and generates selector signals (selC , selN, selS, selE, and selW).

At the heart of the design is the Crossbar switch, which links all inputs and outputs via an adaptable

Fast and accurate packet transmission without interference is made possible by the arbiter's selector signals, which tell the crossbar switch which input to forward to each associated output. The routed data departs through output ports labelled DOC, DON, DOS, DOE, and DOW, each corresponding to a specific direction and destination node within the NoC. Real-time applications on SoCs can benefit from the system's general design, which supports parallel data flow while preserving low latency, high throughput, and minimal area overhead. When taken as a whole, this architecture shows how to effectively address communication constraints at the chip level. integrating modular, scalable buffering and switching methods with sophisticated control logic.

IV PROPOSED SYSTEM

The initial design studied incorporates FIFO, Arbiter, Cross- bar and Buffer. The circuit is shown in Fig. 2 The suggested design provides a high-speed, intelligent NoC router architecture that improves upon existing designs by integrating intelligent buffering using optimized FIFOs, a priority-based arbiter with rotating masking logic, and a dynamic crossbar switch for efficient data routing. While selective FIFO activation minimizes needless switching activity and hence lowers power consumption, the intelligent arbiter guarantees equitable and dynamic bandwidth allocation. This design is appropriate for complicated SoC (System-on-Chip) environments because it dramatically lowers latency, improves throughput, and offers improved scalability. A deep comprehension of important factors like topology, flow management, switching methods, and routing algorithms is necessary while designing a Network-on-Chip (NoC) router.

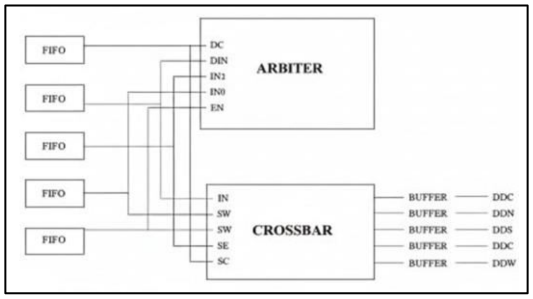


Fig.2 Circuit Diagram

Topology determines how nodes and communication channels are integrated inside the network Over time, a number of topologies, such as mesh, torus, star, spin, and butterfly, have been put forth. Among these, mesh topology is generally selected due to its simplicity, ease of integration, and capacity to assign addresses that correspond well with messages. Low latency, high throughput, low power consumption, low cost, and overall excellent performance should be the goals of an ideal topology. But it's difficult to accomplish all of these objectives at once. Performance is greatly influenced by the switching strategy, which specifies how the router links its inputs and outputs. Either circuit switching or packet switching can be used to accomplish this. While packet switching transmits data packets as soon as they become available without waiting for the entire way to be built up, circuit switching first establishes a dedicated path before any data is delivered.

Each data packet's path from source to destination is decided by routing techniques. Conditions like livelock, deadlock, and hunger should be avoided with an efficient routing method. Similarly, flow control is responsible for managing the allocation of resources for packet transmission. While ineffective flow control methods can cause data packets to become caught in queues, effective flow control techniques guarantee smooth throughput and prevent stalemate. By monitoring the route followed by data packets and how resources are managed along the way, we can analyze and enhance the overall performance of the NoC router.

FIFO:

The oldest request in a queue is treated first when using the FIFO approach. It can be implemented in hardware as a read/write memory or an array of flip-flops that saves data from one clock domain and provides it to other clock domains in line with FIFO logic upon request [11–14]. Read blocks, write blocks, empty blocks, full blocks, memory maps, and counter blocks to counter the two pointers are all included in a FIFO. A FIFO has two data pointers: one for writing to RAM and one for reading from RAM. FIFO first checks the header bit to ensure the existence of data. With the help of a specific port's grant signal, read and write addresses are updated. The data packet that was ejected from the FIFO is transmitted to the appropriate crossbar input, and the destination address of the packet is sent to the arbitrator to carry out the arbitration process. The Arbiter is acknowledged as the control center of the router. Using round-robin arbitration and routing computation, the arbitrator selects five direction ports one at a time.

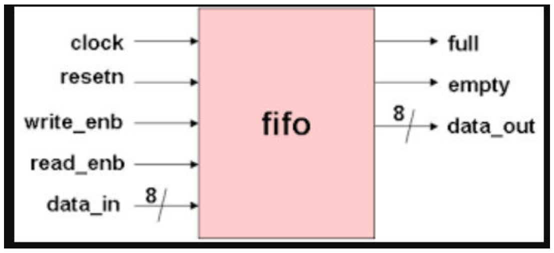


Fig.3 Block Diagram Of FIFO

ARBITER

The router's control center is acknowledged as the Arbiter. The arbiter is used to carry out round robin scheduling method which allows the data from each FIFO buffer for a particular interval of time. In computer science and operating systems, the Round Robin scheduling algorithm is used to distribute CPU time among programs in a time-sharing fashion. Round Robin gets its name from the way the algorithm divides equal time slots for each process and assigns the CPU to each task in a cyclic order [15–18]. This keeps one process from using all of the CPU time and enables each process to get a fair part of it.

In real-time systems where tasks must be completed within a set amount of time, the Round Robin algorithm is frequently utilized. It offers an easy and effective technique to distribute resources, avoid starvation, and enhance system efficiency.

EN	in [0]	in [1]	in [2]	in [3]	output [0]	output [1]	output [2]	output [3]
0	X	X	X	X	0	0	0	0
1	1	X	X	X	1	0	0	0
1	0	1	X	X	0	1	0	0
1	0	0	1	X	0	0	1	0
1	0	0	0	1	0	0	0	1

Table.2 Truth Table

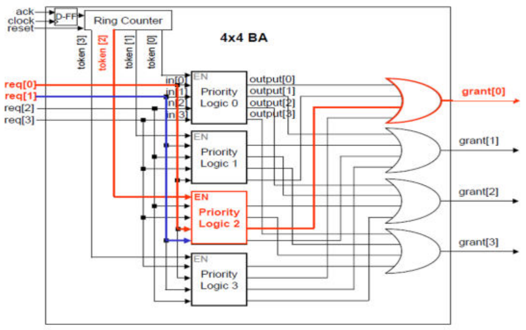


Fig.4 Circuit Diagram

CROSS BAR

A crossbar is a module that combines muxes and demuxes. There is a connection between an input and an output port. There is no feedback in this design. Crossbar can only create one link at a time. From inputs of Cin, Ein, Nin, Sin, and Win, binary outputs are Cout, Eout, Nout, Sout, and Wout. The output and input are both 8-bit. Four binary combinations are produced by reducing the select line in this design to two lines.

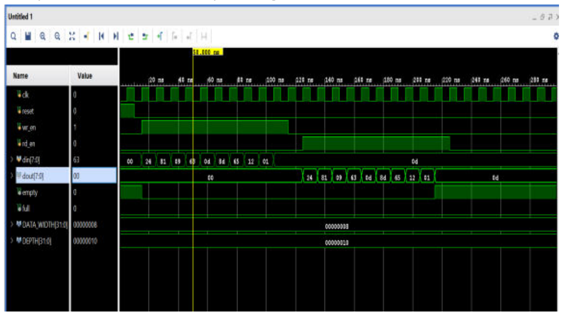
Input port	*Select*	*Output Port*
SC	00	CNI
SC	01	CWI
SC	10	CEI
SC	11	CSI
SN	00	NWI
SN	01	NEI
SN	10	NSI
SN	11	NCI
SS	00	SCI
SS	01	SNI
SS	10	SWI
SS	11	SEI
SE	00	ESI
SE	01	ECI
SE	10	ENI
SE	11	EWI
SW	00	WEI
SW	01	WSI
SW	10	WCI
SW	11	WNI

Table.2. Truth Table for Cross Bar operation

Once each of these blocks has been designed separately using structural modelling, they are all merged and simulated. The power consumption rating for this router design is determined using the XPower analyser. Buffers will aid in lowering latency, according to the paper [15]. As a result, the design is altered to include an extra buffer at each router output port. The design is enhanced with more buffers at the output ports. The block diagram of the new design is shown in Fig. 2 and is implemented in Xilinx. The buffers are seen at the output ports of the schematic.

V IMPLEMENTATION AND PERFORMANCE EVALUATION

The Kintex 7 FPGA board's Xilinx software is used to implement the design. The board belongs to the XC7K70T series. The board has total of 82,000 slice registers, 41,000 slice LUT's,328 LUT-FF pairs, 320 Bounded IOB's. The comparison table of device utilisation is shown in Table 2, which includes information on the quantity of a certain component used. The comparison of power consumption, timing analysis, and memory usage is shown in Table 3.



Name	Value
clk	0
reset	0
wr_en	1
rd_en	0
din[7:0]	63
dout[7:0]	00
empty	0
full	0
DATA_WIDTH[31:0]	00000008
DEPTH[31:0]	00000010

Fig.5 FIFO SIMULATION

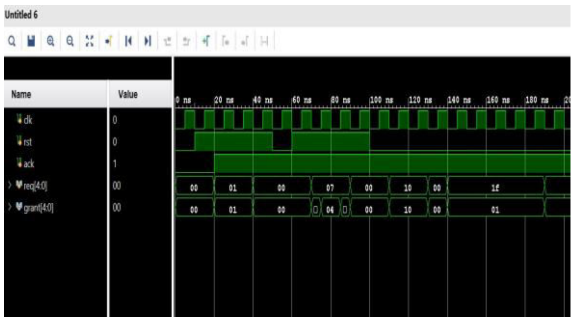


Fig.6 ARBITER SIMULATION

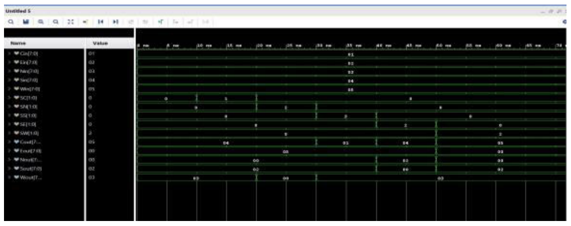


Fig.7 CROSSBAR SWITCH

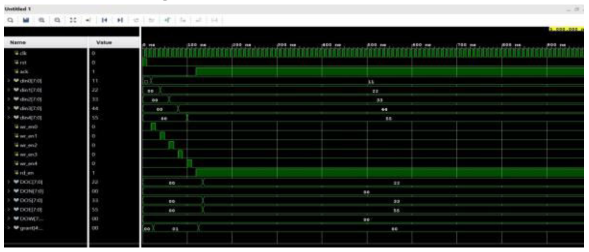


Fig.8 NoC SIMULATION

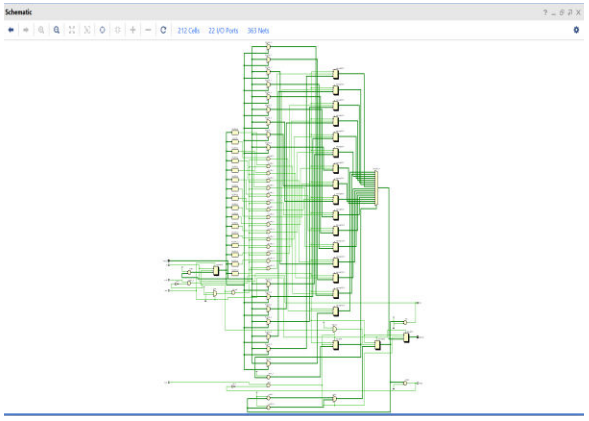


Fig.9 FIFO SCHEMATIC

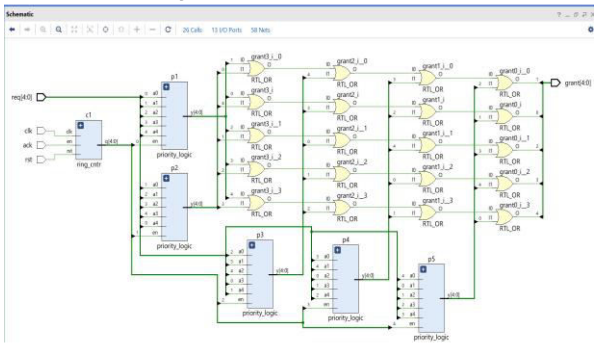


Fig.10 ARBITARY SCHEMATIC

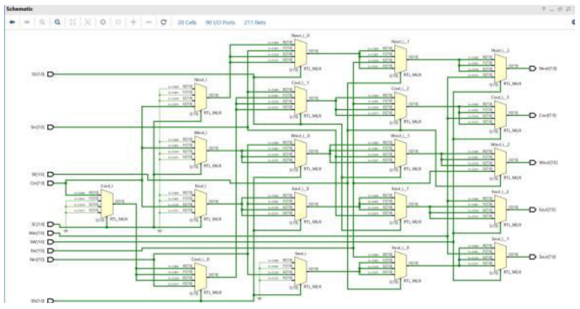


Fig.11 CROSSBAR SWITCH SCHEMATIC

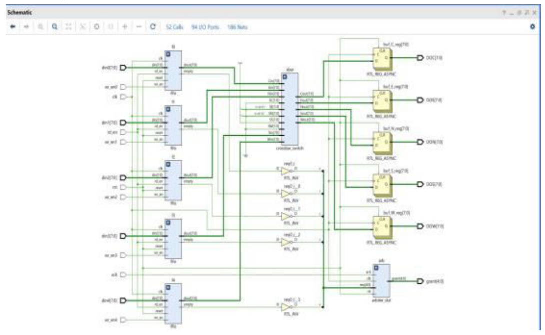
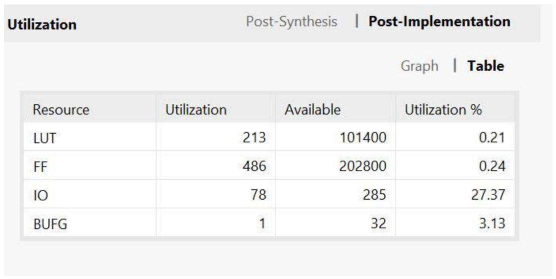


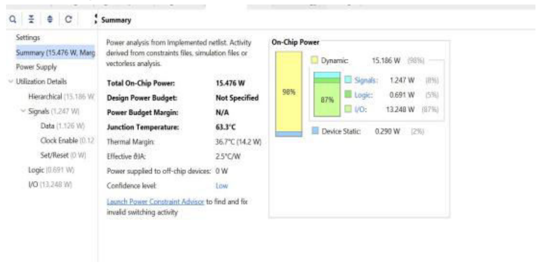
Fig.12 NoC SCHEMATIC


Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	213	101400	0.21
FF	486	202800	0.24
IO	78	285	27.37
BUFG	1	32	3.13

Fig.13 Area Utilization of NOC Router


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

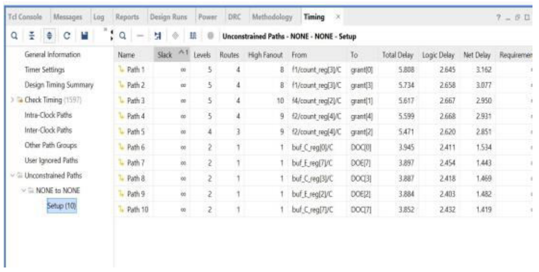
Total On-Chip Power:	**15.476 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**63.3°C**
Thermal Margin:	36.7°C (14.2 W)
Effective θJA:	2.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	15.186 W	(98%)
Signals:	1.247 W	(8%)
Logic:	0.691 W	(5%)
I/O:	13.248 W	(87%)
Device Static:	0.290 W	(2%)

Fig.14 Power Results



Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	5	4	8	f1/count_reg[3]/C	grant[0]	5.808	2.645	3.162
Path 2	∞	5	4	8	f1/count_reg[3]/C	grant[3]	5.734	2.658	3.077
Path 3	∞	5	4	10	f4/count_reg[2]/C	grant[1]	5.617	2.667	2.950
Path 4	∞	5	4	9	f2/count_reg[4]/C	grant[4]	5.599	2.668	2.931
Path 5	∞	4	3	9	f2/count_reg[4]/C	grant[2]	5.471	2.620	2.851
Path 6	∞	2	1	1	buf_C_reg[0]/C	DOC[0]	3.945	2.411	1.534
Path 7	∞	2	1	1	buf_E_reg[7]/C	DOE[7]	3.897	2.454	1.443
Path 8	∞	2	1	1	buf_C_reg[3]/C	DOC[3]	3.887	2.418	1.469
Path 9	∞	2	1	1	buf_E_reg[2]/C	DOE[2]	3.884	2.403	1.482
Path 10	∞	2	1	1	buf_C_reg[7]/C	DOC[7]	3.852	2.432	1.419

Fig.15 Delay Result

Table-Compressions Of existing and proposed method

	Area(Luts)	Power(Micro watts)	Delay(Max)ns	Dealy(Min)ns
Existing Method	213	7.808	5.15	0.3
Proposed Method	213	15	4.75	0.2

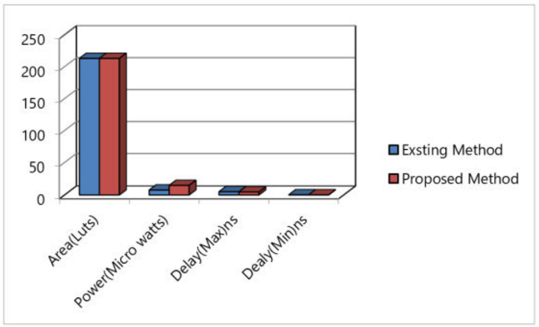


Fig.16 Graph comparison

VI CONCLUSION

This project successfully presents a high-speed VLSI router architecture designed to improve data transmission within Network-on-Chip (NoC) systems. The design minimizes data congestion, lowers latency, and increases throughput by incorporating FIFO-based intelligent buffering at input ports. The inclusion of a dynamic priority-based arbiter and a flexible crossbar switch allows for efficient path selection and scalable routing. Simulation and testing using the Vivado Design Suite confirmed functional correctness and demonstrated the router's ability to handle dynamic traffic scenarios reliably. The modular, synthesizable design balances area usage and performance, making it well-suited for real-time applications in advanced SoCs such as AI accelerators, multimedia processors, and embedded platforms.

REFERENCES

[1] T. A. C. M. Claasen, "An industry perspective on current and future state of the art in system-on-chip (SoC) technology," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1121–1137, Jun. 2006, doi: 10.1109/JPROC.2006.873616.

[2] G. Martin and H. Chang, "System-on-chip design," in *Proc. 4th Int. Conf. ASIC (ASICON)*, Shanghai, China, 2001, pp. 12–17, doi: 10.1109/ICASIC.2001.982487.

[3] I. Giechaskiel and J. Szefer, "Information leakage from FPGA routing and logic elements," in *Proc. 39th Int. Conf. Computer-Aided Design (ICCAD)*, 2020.

[4] M. Katta and T. K. Ramesh, "Latency improvement by using FILL VC allocation for network on chip," in *Proc. Int. Conf. Data Engineering and Communication Technology (ICDECT)*, Warangal, India, 2020.

[5] N. Varghese and R. Swaminadhan, "Power efficient router architecture for scalable NoC," in *Innovations in Electronics and Communication Engineering*, Singapore: Springer, 2020.

[6] X. Wang, T. Ahonen, and J. Nurmi, "A synthesizable RTL design of asynchronous FIFO," in *Proc. Int. Symp. System-on-Chip*, 2004.

[7] M. M. Wanjari, P. Agrawal, and R. V. Kshirsagar, "Design of NoC router architecture using VHDL," *International Journal of Computer Applications*, vol. 115, no. 4, 2015.

[8] T. Balharith and F. Alhaidari, "Round robin scheduling algorithm in CPU and cloud computing: A review," in *Proc. 2nd Int. Conf. Computing*, 2019.

[9] G. Verma et al., "Design and implementation of router for NoC on FPGA," *International Journal of Future Generation Communication and Networking*, vol. 9, no. 12, pp. 263–272, 2016.

[10] N. L. Venkataraman, R. Kumar, and P. M. Shakeel, "Ant lion optimized bufferless routing in the design of low power application specific network on chip," *Circuits, Systems, and Signal Processing*, vol. 39, no. 2, pp. 961–976, 2020.

[11] S. Madhavan and H. P. V, "Design and verification of 1×5 router," in *Proc. IEEE Mysuru Sub Section Int. Conf. (MysuruCon)*, Mysuru, India, 2022, pp. 1–6.

[12] J. A. Williams, N. W. Bergmann, and X. Xie, "FIFO communication models in operating systems for reconfigurable computing," in *Proc. 13th Annual IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, 2005, pp. 277–278, doi: 10.1109/FCCM.2005.35.

[13] J. T. Han, Y. Guan, and Z. Dai, "Implementation of SoC-PC communication interface based on USB2.0," in *Proc. Int. Conf. New Trends in Information and Service Science*, Beijing, China, 2009, pp. 831–834, doi: 10.1109/NISS.2009.53.

[14] M. Oveis-Gharan and G. N. Khan, "Index-based round-robin arbiter for NoC routers," in *Proc. IEEE Computer Society Annual Symp. VLSI (ISVLSI)*, Montpellier, France, 2015, pp. 62–67, doi: 10.1109/ISVLSI.2015.27.

[15] A. Mangukia et al., "Improved variable round robin scheduling algorithm," in *Proc. 12th Int. Conf. Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2021, pp. 1–7, doi: 10.1109/ICCCNT51525.2021.9579716.

[16] W. Ullah and M. A. Shah, "A novel resilient round robin algorithm based CPU scheduling for efficient CPU utilization," in *Competitive Advantage in the Digital Economy (CADE)*, Venice, Italy, 2022, pp. 41–48, doi: 10.1049/icp.2022.2038.

[17] S. M. Hassan and S. Yalamanchili, "Centralized buffer router: A low latency, low power router for high radix NoCs," in *Proc. IEEE/ACM Int. Symp. Networks-on-Chip (NoCS)*, 2013.

[18] B. Zhao, Y. Zhang, and J. Yang, "A speculative arbiter design to enable high-frequency many-VC router in NoCs," in *Proc. IEEE/ACM Int. Symp. Networks-on-Chip (NoCS)*, Tempe, AZ, USA, 2013, pp. 1–8, doi: 10.1109/NoCS.2013.6558415.

[19] G. Xiaopeng, Z. Zhe, and L. Xiang, "Round robin arbiters for virtual channel router," in *Proc. Multiconference on Computational Engineering in Systems Applications (CESA)*, Beijing, China, 2006, pp. 1610–1614, doi: 10.1109/CESA.2006.4281893.

[20] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Automation Conference (DAC)*, Las Vegas, NV, USA, 2001, pp. 684–689, doi: 10.1145/378239.379001.

[21] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002, doi: 10.1109/2.976921.