Hindawi
International Journal of Reconfigurable Computing
Volume 2019, Article ID 7239858, 9 pages
https://doi.org/10.1155/2019/7239858

Research Article

AsyncBTree: Revisiting Binary Tree Topology for Efficient FPGA-Based NoC Implementation

Kizheppatt Vipin

Department of Electrical and Computer Engineering, Nazarbayev University, Astana, Kazakhstan

Correspondence should be addressed to Kizheppatt Vipin; vipin.kizheppatt@nu.edu.kz

Received 29 October 2018; Revised 11 January 2019; Accepted 7 February 2019; Published 20 February 2019

Academic Editor: John Kalomiros

Binary tree topology generally fails to attract network on chip (NoC) implementations due to its low bisection bandwidth. Fat trees are proposed to alleviate this issue by using increasingly thicker links to connect switches towards the root node. This scheme is very efficient in interconnected networks such as computer networks, which use generic switches for interconnection. In an NoC context, especially for field programmable gate arrays (FPGAs), fat trees require more complex switches as we move higher in the hierarchy. This restricts the maximum clock frequency at which the network operates and offsets the higher bandwidth achieved through using fatter links. In this paper, we discuss the implementation of a binary tree-based NoC, which achieves better bandwidth by varying the clock frequency between the switches as we move higher in the hierarchy. This scheme enables using simpler switch architecture, thus supporting higher maximum frequency of operation. The effect on bandwidth and resource requirement of this architecture is compared with other FPGA-based NoCs for different network sizes and traffic patterns.

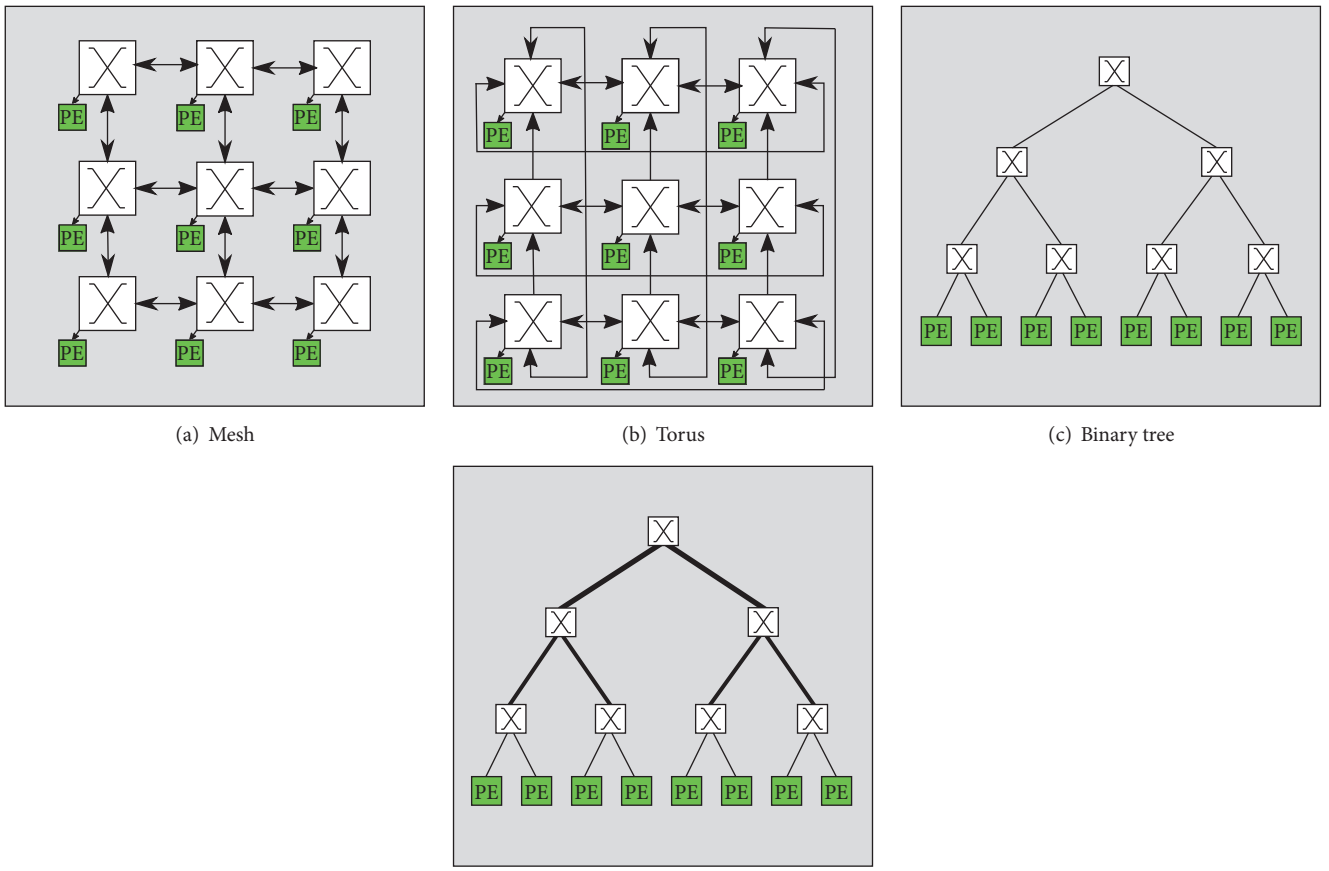
1. Introduction

Network on chip (NoC) architectures enable high-performance, scalable, and power-efficient multicore systems for modern compute and communication intensive applications [1, 2]. Researchers have proposed different NoC topologies such as mesh, ring, torus, binary trees, and star, each having varying degrees of quality of service, bandwidth, and latency [3, 4]. Despite their simple architecture and routing algorithms, binary trees are generally not attractive for NoC implementations. It is mainly because of their lower bisection bandwidth. Fat trees are proposed as a remedy to improve the bandwidth by adding more number of links when moving towards the root node. This solution works well in traditional interconnect networks such as computer networks [5]. In an NoC environment, their advantage is limited since the switch complexity increases as the network size increases. This limits the maximum supported clock frequency of the network, thus bringing down the system performance.

Theoretically, instead of increasing the link width between tree levels, increasing the clock frequency between them should provide the same benefit. In traditional computer networks, this may not be possible since the network interfaces operate on predefined standards, which restrict the frequency of operation. In an NoC environment, this is very much possible since all the compute instances are within the same chip and are not restricted by any physical protocol. Modern FPGAs support asynchronous FIFOs, which make the implementation of such asynchronous switches easier. These switches have simpler architecture than fat tree switches and support better clock frequency. But they are more resource-intensive compared to traditional binary trees.

Although asynchronous NoCs were proposed previously for integrated circuits, they are not evaluated for binary tree performance improvement [6–8]. A quantitative analysis is missing in the literature, especially for FPGA implementations. In this work, we present a quantitative analysis of different tree topologies, namely, the binary tree, binary fat tree, and asynchronous binary tree when targeting FPGA-based NoC implementation. The main contributions of this work are

(i) detailed design of an open-source globally asynchronous-locally synchronous (GALS) binary



(a) Mesh (b) Torus (c) Binary tree

(d) Fat tree

Figure 1: Different NoC topologies.

tree-based NoC implementation targeting Xilinx FPGAs,

(ii) performance evaluation of the proposed infrastructure with traditional binary trees and the state-of-the-art open-source binary fat tree, torus, and mesh topologies,

(iii) an analysis of trade-off points for the different implementations when targeting FPGAs.

The remainder of this paper is organized as follows: Section 2 discusses the relevant background, Section 3 discusses the architecture of the proposed NoC, Section 4 discusses the performance metrics, and Section 5 concludes the paper and gives the future research directions.

2. Background

Network on chip is an interconnect approach that helps different IPs and subsystems in a chip to communicate with each other in an efficient and scalable manner. In this approach, each processing element (PE) is connected to a switch and multiple switches are interconnected to form a network. A PE could be a processor core, a DSP core, or an IP block. The network infrastructure helps in routing data from one PE to another in the form of data packets. NoCs have found varying applications such as image and signal processing [1], multiprocessor systems [9], and virtual machine implementations [10]. Based on how the switches are interconnected, there are different NoC topologies such as mesh, torus, tree, ring, star, and BFT, as shown in Figure 1 [11].

In a mesh topology, every switch, except the ones on the edges, is connected to 4 other neighboring switches. A torus topology is similar to mesh but is cyclic in nature. In a binary tree, switches are arranged in a hierarchy. Each switch has a parent node and two child nodes. Unlike mesh and torus, where each switch has a corresponding PE, in a tree topology only the switches at the bottom most level (leaf nodes) are connected to PEs.

For interconnected networks, an important performance parameter is the bisection bandwidth [12]. It is defined as the minimum bandwidth between two equal partitions of the network. For a mesh topology, it is $\sqrt{n} * B$, where n is the number of switches and B is the bandwidth of a single link. For torus, it is twice that of mesh, but for a binary tree, it is only B. To address this issue, instead of using a single link between switches, more links can be used between them as we go higher in the tree hierarchy. Such topology is called a fat tree [13, 14]. Although this will improve the bisection bandwidth, the switches in the upper hierarchy

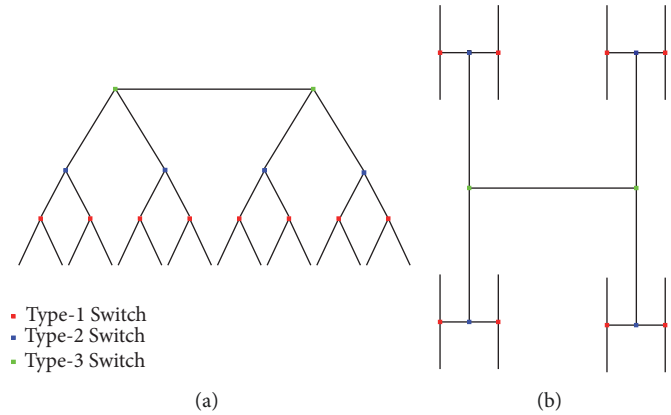


Figure 2: (a) A binary tree topology utilizing switches operating at different clock frequencies. (b) The tree as an H-tree for better floorplanning on the FPGA.

become more and more complex. In this work, we analyze whether using asynchronous switches with same link width can provide similar performance of fat trees while keeping relatively simpler switches.

Æthereal is a popular NoC implementation, which provides guaranteed quality of service through pipelined time-division-multiplexed circuit switching [15]. But it requires to explicitly set up a channel on the routing path before transmitting the first payload packet, and a flow cannot use more than its guaranteed bandwidth share even if the network is underutilized. The MANGO [16] architecture is a clock-less NoC to provide connection-oriented guaranteed services (GS) as well as connection-less best-effort (BE) routing. The clock-less implementation has the advantage of zero dynamic power consumption when the network is idle, but the main challenge is its interfacing with the standard IP cores. IP cores available from vendors and third-party developers are synchronous in nature, thus requiring a clock signal between them and the NoC. Both these implementations target ASIC implementations and are not evaluated on FPGAs to understand the maximum possible speed of their operation.

There have been previous efforts to develop NoC architectures specifically targeting FPGAs. CONNECT NoC generator is the most popular among them [17]. CONNECT is inspired by the fact that FPGAs have a large routing infrastructure available compared to memory and logic elements and tries to exploit it. It supports different NoC topologies and uses a single-stage pipeline mechanism to minimize hardware and latency. It has low operating frequency and is still quite resource-intensive as seen in Section 4. Split-merge is another NoC infrastructure developed at University of Pennsylvania [18]. It tries to overcome the limited clock performance of CONNECT at the expense of few more resources.

To the best of our knowledge, there have been no previous efforts to implement asynchronous switch-based NoCs targeting FPGAs. In this work, we give a quantitative analysis of the performance of asynchronous binary trees. We compare their performance with traditional and fat trees as well as other popular FPGA NoCs. The binary tree implementations are available as open source for other researchers to verify and to improvise the designs.

3. Architecture

An AsyncBTree (asynchronous binary tree) tries to achieve better performance compared to a conventional binary tree and binary fat tree in terms of resource utilization and throughput by applying topology specific optimizations and using asynchronous links between different tree levels. Figure 2 shows the architecture of an AsyncBTree utilizing different kind of optimized switches at different levels. The detailed architectures of the switches are discussed in Section 3.1. The binary tree is placed and routed in the FPGA as an H-tree for efficient resource utilization and better floorplanning. Such placement also supports partial reconfiguration of a portion of the NoC in an efficient manner.

As the first optimization, the root node of the tree is removed and the switches at level-1 are directly connected. Since the connectivity of the root node is only 2, in practical systems, they act as a transparent switch. They could be useful where packets are injected to the NoC through the root node by making its connectivity into 3. For FPGAs, external interfaces such as PCI express and Ethernet are used for injecting packets. The hard macros corresponding to these interfaces are situated along the periphery of the chips. Thus, they will provide better clock performance when the packets are injected from one of the leaf nodes which incorporates one of these hard macros. Removing the root node helps in reducing the resource utilization.

3.1. Switches. AsyncBTrees use three different kinds of switches for packet routing. The architecture of the proposed type-1 switch is as shown in Figure 3(a). Type-1 switches are used only in the leaf nodes for directly interfacing with PEs. The switch has separate interface for receiving and transmitting packets from two PEs and a single interface to the parent node. Each receive and transmit interface to/from the PEs is connected to asynchronous FIFOs. The interface to the parent node implements a single synchronous FIFO for the receive interface but no transmit FIFO. Thus, each switch contains 5 FIFOs.

Asynchronous FIFOs can operate their read and write interfaces using independent clocks. Depth of the FIFOs is

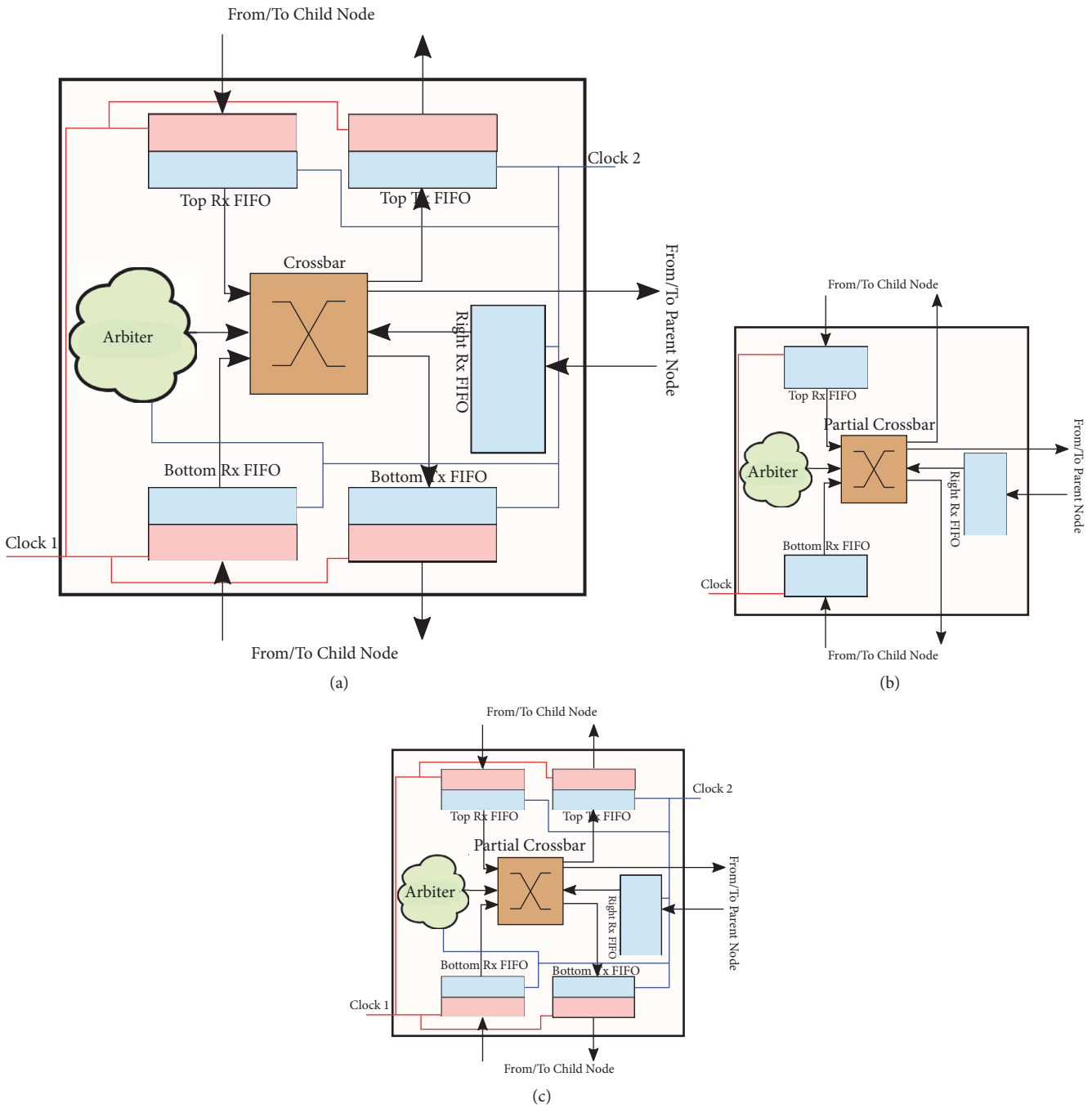


Figure 3: Architecture of the different switches used in the AsyncBTree. (a) Type-1 switch used with the leaf nodes (PEs), with complete crossbar switch and asynchronous FIFOs with receive and transmit interfaces. (b) Type-2 switch used in intermediate nodes with partial crossbar switch and synchronous FIFOs. (c) Type-3 switches used in intermediate switches with partial crossbar and asynchronous FIFOs with receive and transmit interfaces.

kept very low (16) to reduce the resource utilization. The asynchronous FIFOs receive data from the downstream ports on Clock 1 signal. The received packets are routed to the appropriate output ports by an arbitrator through a crossbar switch. The read side of the receive FIFO, the write side of the transmit FIFO, the arbitrator, and the crossbar work on Clock 2 signal, whose frequency will be much higher than that of Clock 1. In order to match the performance of a binary fat tree, Clock 2 frequency should be twice as that of Clock 1. Type-1 switches implement a full crossbar, which enables loop-back of packets at the switch level. The arbitrator internally uses flit-level round-robin arbitration scheme to select the input port when more than one port requests the same output port.

Figure 3(b) shows the architecture of a type-2 switch. Type-2 switches are synchronous in nature and are similar to the switches of traditional binary trees. These switches

implement only partial crossbar switch where an incoming packet cannot be routed back to the same port. Since these switches are used only in intermediate levels, there is no necessity to support loop-back since they are already implemented by type-1 switches. This simplifies the switch design and helps in reducing resource utilization.

Type-3 switches (Figure 3(b)) are similar to type-1 switches except that they implement only a partial crossbar similar to type-2 switches. Theoretically, to match the performance of a binary fat tree, AsyncBTrees should be implementing type-3 switches at every level with increasing clock frequencies. But, in practical scenarios, doubling clock frequency at each level is not possible in FPGAs as the tree size increases. The maximum frequency supported by modern devices is in the order of hundreds of megahertz. Hence, for practical implementations, AsyncBTrees increase the clock frequency only for alternative levels. For example, for a NoC with 8 PEs and three levels (as shown in Figure 2), clock frequency is doubled between the PE interface and the output of level-3 (type-1) switches. The input and output frequencies of next-level switches (here type-2 switches) are the same, which are equal to the output frequency of type-1 switches. Again, the clock frequency is doubled between the input and output of level-1 switches (here type-3 switches). Although asynchronous FIFOs can operate using same clock signal for read and write interfaces, the resource utilization of asynchronous FIFOs is much more than its synchronous counterpart for a given FIFO size. For reducing resource consumption, levels which operate on synchronous clock use type-2 switches and levels which operate on asynchronous clock use type-3 switches.

3.2. Packet Format. The NoC uses a simple packet format with destination PE address and payload as shown in Figure 5. The NoC implementation is fully configurable such that it can support any data width with varying number of PEs. The total packet size depends upon these two parameters.

3.3. Routing. AyncBTree uses fixed routing based on the destination address of the packet header. The routing is flit-level, meaning each packet is expected to have the destination PE address in the header. Larger packets are sent as multiple flits. One major advantage of binary trees is that the multiple packets sent from one PE to another will be always delivered in the sent order. In other packet switched networks such as mesh or torus, the packets could be delivered out of order depending on the routing algorithms. In this case, additional logic is required for packet reassembly and packet numbers also have to be inserted into the payload.

The routing table of each switch in AsyncBTree contains four entries corresponding to the smallest and largest PE addresses in its left and right subtrees. If the destination address is within the range of left subtree, it is routed left and if it is within the range of right subtree, the packet is sent right. If the address is not within these ranges, the packet is routed towards the parent node. Due to the deterministic routing policy and since the routing is at flit-level, the NoC is free of dead or live locks.

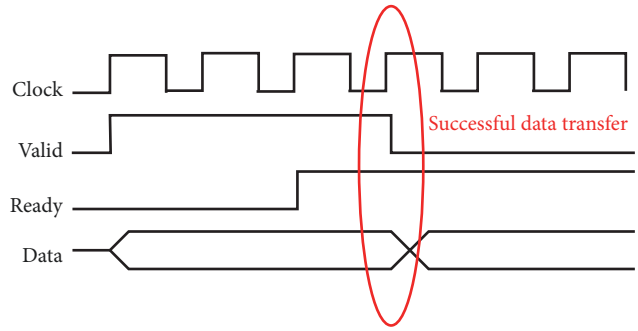


Figure 4: AXI4-Stream protocol.

3.4. Flow Control. The current AsyncBTree implementation does not include virtual channels and flow control is achieved at electrical signaling level. The interface between switches as well as between switches and PEs confirms to AXI4-Stream interface. Such interface also enables seamless integration of several vendor-supported IP cores directly with the NoC. As per AXI4-Stream protocol, a successful data transfer happens only when the *valid* signal from the transmitter and *ready* signal from the receiver are asserted simultaneously as shown in Figure 4.

For comparison purpose, following the same design principles, we implemented traditional (synchronous) binary trees also. They follow the same arbitration and routing architecture except that all asynchronous FIFOs are replaced with synchronous counterparts.

4. Results and Discussion

In this section, we discuss the implementation and performance comparison of the different NoC implementations. All designs are modeled using Verilog HDL and extensively simulated for their functional correctness. We use the CONNECT open-source NoC platform as the fat tree, mesh, and torus references [17]. The designs are simulated as well as implemented with Vivado 2017.3 targeting Xilinx xc7vx690t FPGA (VC709 evaluation board).

Table 1 compares the resource utilization and the maximum frequency of operation for the binary tree, AsyncBTree, and fat tree for different network sizes (number of PEs). For all implementations, the interface between PEs and the switches is kept 32-bits wide. As expected, binary trees are least resource-intensive due to their simple switch architecture. AsyncBTree consumes 45% to 65% more LUTs (look-up-tables) and about 165% more flip-flops compared to the binary tree implementation. The multiple frequencies in the AsyncBTree rows represent the maximum clock frequency supported at different tree levels. At the lowest level (switches connected to PEs), the clock performance is better than that of binary trees but deteriorates as it goes higher in the hierarchy. This could be because of the additional pipelining present inside the asynchronous FIFOs. This also means if AsyncBTree is used as a synchronous NoC (all tree levels are clocked by a single clock source), its resource consumption and performance will be worse than a binary tree.

Figure 5: Packet structure.

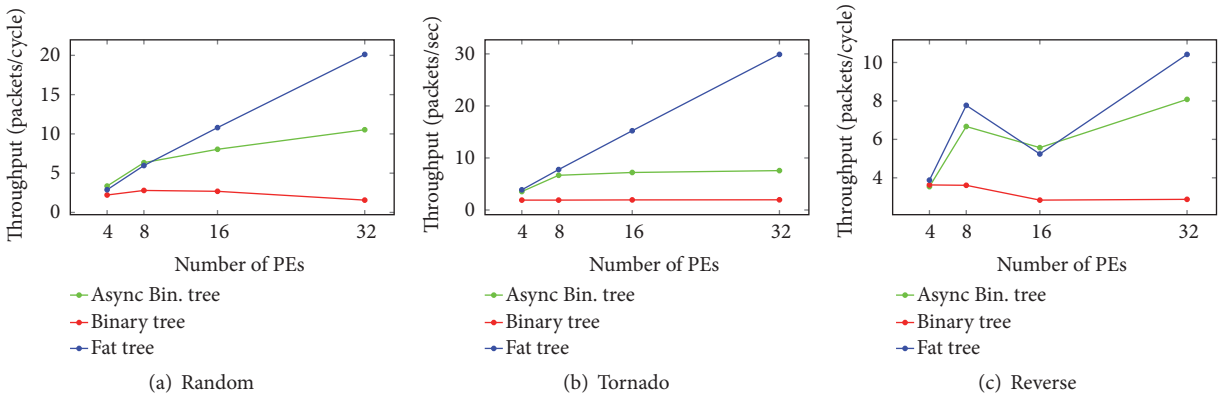


Figure 6: Throughput of different binary NoC architectures with varying size corresponding to different traffic patterns when all of them are clocked at the same clock frequency.

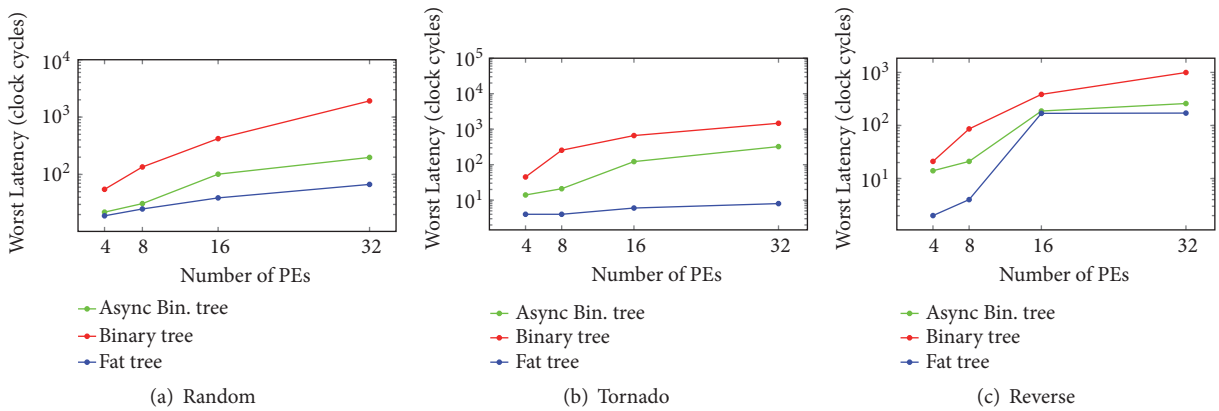


Figure 7: Worst-case latency of different binary NoC architectures with varying size corresponding to different traffic patterns when all of them are clocked at the same clock frequency.

Compared to AsyncBTree, fat trees consume ~3.7× LUTs but less than half the number of flip-flops. AsyncBTree requires more flip-flops due to the presence of asynchronous FIFOs. For fat trees, the impact due to complex switches can be clearly seen in the clock performance, where they are not even able to achieve 200 MHz for a high-end FPGA. Due to the low clock performance of the NoC, the PEs also have to be underclocked in most scenarios for overall synchronous operations. Considering the fact that the number of LUTs available in 7-series Xilinx FPGAs is half of that of number of flip-flops, AsyncBTree is much lite compared to fat trees at the same time given more than 4× clock performance.

Table 2 lists the resource utilization and clock performance of the most popular FPGA NoC topologies, namely, mesh and torus. Data shows that these topologies are quite resource-intensive compared to all binary tree configurations, especially the number of LUTs. In terms of clock performance, for larger NoC configurations, they perform better than fat trees but are inferior to traditional binary trees and the proposed AsyncBTrees.

Figures 6 and 7 compare the throughput and latency of the three implementations with different NoC sizes for different traffic patterns such as random, tornado, and reverse [19]. The different patterns are generated based on how the destination addresses are generated for each data packet. In each case, the PE to switch interface is clocked at the lowest frequency supported among the three implementations. For AsyncBTree, upper levels are clocked at double the frequency of lower levels but is limited by maximum supported frequency given in Table 1. It could be seen that, for NoC size up to 8 PEs, AsyncBTree performance is better than or comparable to that of fat trees. For larger tree sizes, fat trees perform better, since the clock frequencies cannot be scaled beyond a limit. If PEs run at lower clock frequencies (~50 MHz), AsyncBTree can provide better performance for NoC with up to 16 PEs.

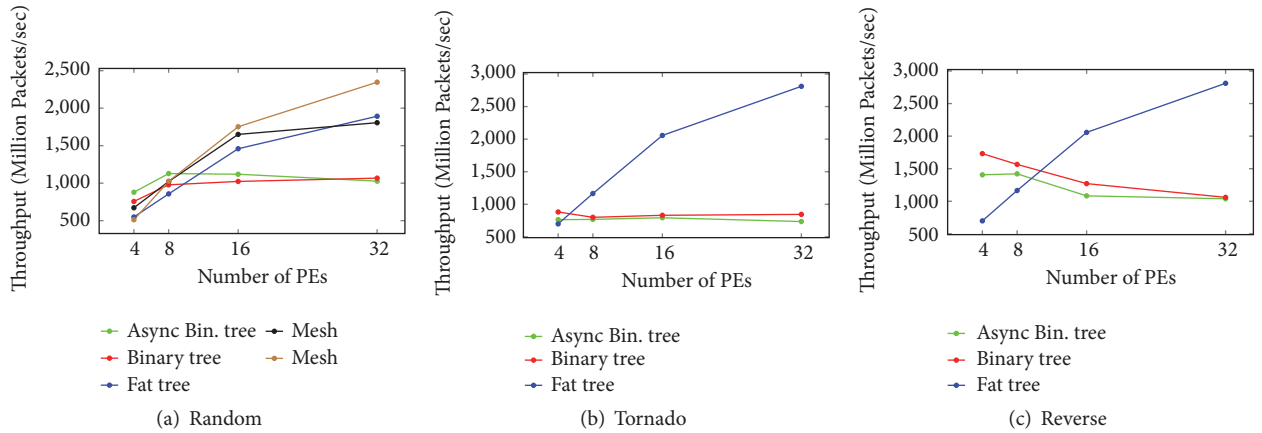


Figure 8: Throughput of different NoC architectures with varying size corresponding to different traffic patterns when all of them are clocked at the maximum supported clock frequency.

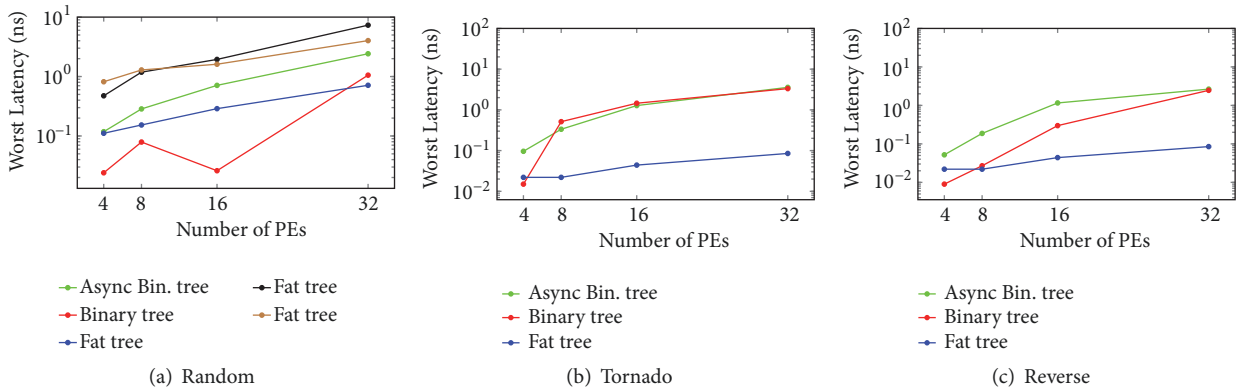


Figure 9: Maximum latency of different binary NoC architectures with varying size corresponding to different traffic patterns when all of them are clocked at the maximum supported clock frequency.

Table 1: Resource utilization and maximum clock performance of different binary tree based NoC configurations.

NoC	# of PEs	LUTs	FFs	Fmax (MHz)
Fat Tree	4	2833	760	180
Fat Tree	8	3660	912	150
Fat Tree	16	12431	3040	135
Fat Tree	32	37047	8512	94
Binary Tree	4	522	510	450
Binary Tree	8	1616	1566	407
Binary Tree	16	3603	3726	420
Binary Tree	32	6077	6850	425
AsyncBTree	4	760	1350	481,416
AsyncBTree	8	2242	4100	506,407,418
AsyncBTree	16	4212	7506	483,386,416
AsyncBTree	32	10046	18360	460,387,377

Figures 8 and 9 compare the throughput and latency when each implementation is running at its maximum supported frequency. Again, for smaller NoC sizes, binary tree and AsyncBTree outperform fat tree for random traffic pattern. But, for larger NoC sizes, fat trees are clearly advantageous. Figure 8(a) also shows the performance of mesh topology, which is the most popular NoC topology, compared to different tree topologies. Again, for smaller NoC sizes (8 or less), AsyncBTree's performance is better. Several FPGA-based multiprocessor systems have 8 cores or less; thus AsyncBTree could be more suitable for their implementation.

Figure 10 compares the performance of each NoC compared to its resource utilization. The total number of resources is calculated by adding the number of LUTs with the scaled number of flip-flops. The number of flip-flops is multiplied by a factor of 0.5 since in Xilinx 7-series FPGAs there are twice the number of flip-flops compared to LUTs in every logic slice. Synchronous binary trees clearly have an upper hand in this regard. AsyncBTree gives moderately high throughput by consuming relatively less resources. But, for larger NoC size, mesh topology is still the suitable candidate.

5. Conclusion

In this paper, we discussed the implementation of an asynchronous binary tree-based NoC architecture targeting

Table 2: Resource utilization and maximum clock performance of mesh and torus NoC topologies with different configurations.

NoC	# of PEs	LUTs	FFs	Fmax (MHz)
Torus	4	3763	960	164
Torus	8	7615	1920	164
Torus	16	15365	3840	149
Torus	32	32689	7680	141
Mesh	4	1337	576	215
Mesh	8	3940	1344	179
Mesh	16	10471	3072	165
Mesh	32	23798	6528	158

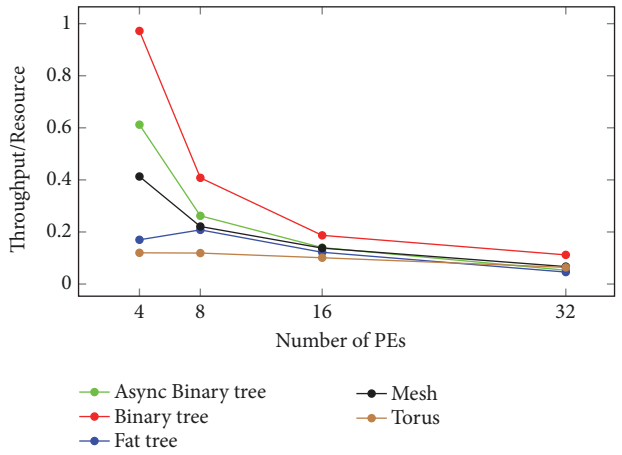


Figure 10: Throughput versus cost.

FPGA implementation and compared it with other tree and mesh implementations. It could be seen that, for low-performance applications, binary trees are best suitable due to their high throughput to resource utilization ratio. For high-performance large NoC sizes, mesh topology is the ideal candidate. For high-performance small NoC sizes, AsyncBTree is a suitable candidate. In future, we will be analyzing the effect of using asynchronous switches with other topologies such as mesh and torus. Implementations of AsyncBTree and binary tree are provided as open source, which could complement the CONNECT NoC platform [20].

Data Availability

All the source code for the designs used for generating the data is available as open source in the following Github Repository under MIT License. The Repo link is https://github.com/vipinkmenon/HNoC.

Conflicts of Interest

The author declares that there are no conflicts of interest.

References

[1] J. Joshi, K. Karandikar, S. Bade, M. Bodke, R. Adyanthaya, and B. Ahirwal, "Multi-core image processing system using network on chip interconnect," in *Proceedings of the 2007 50th Midwest Symposium on Circuits and Systems*, August 2007.

[2] C. Neeb, M. J. Thul, and N. Wehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 1766–1769, May 2005.

[3] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc, San Francisco, Calif, USA, 2003.

[4] S. Kumar, A. Jantsch, J.-P. Soininen et al., "A network on chip architecture and design methodology," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 105–112, USA, April 2002.

[5] G. Shainer, "Networks: topologies how to design," Tech. Rep., HPC Advisory Council, 2011.

[6] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proceedings of the Design, Automation and Test in Europe*, pp. 1226–1231, March 2005.

[7] E. Beigne, P. Vivet, M. Renaudin, and J. Quartana, "Communication node architecture in a globally asynchronous network on chip system," U.S. Patent US7 940 666B2, 2011.

[8] I. M. Panades and A. Greiner, "Bi-synchronous fifo for synchronous circuit communication well suited for network-on-chip in gals architectures," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pp. 83–94, May 2007.

[9] D. Bertozzi, A. Jalabert, S. Murali et al., "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.

[10] G. Mathias and K. Kent, "An embedded java virtual machine using network-on-chip design," in *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping, RSP 2006*, June 2006.

[11] M. Ortín-Obón, D. Suárez-Gracia, M. Villarroya-Gaudó, C. Izu, and V. Viñals-Yúfera, "Analysis of network-on-chip topologies for cost-efficient chip multiprocessors," *Microprocessors and Microsystems*, vol. 42, pp. 24–36, 2016.

[12] W.-C. Tsa, Y.-C. Lan, Y.-H. Hu, and S.-J. Chen, "Networks on chips: structure and design methodologies," *Journal of Electrical and Computer Engineering*, vol. 2012, Article ID 509465, 15 pages, 2012.

[13] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[14] S. R. Ohring, M. Ibel, M. J. Kumar, and S. K. Das, "Generalized fat trees," in *Proceedings of the IEEE 9th International Parallel Processing Symposium*, pp. 37–44, April 1995.

[15] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *IEEE Design Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[16] T. Bjerregaard and J. Sparso, "Implementation of guaranteed services in the MANGO clockless network-on-chip," *IEE Proceedings - Computers and Digital Techniques*, vol. 153, no. 4, pp. 217–229, 2006.

[17] M. K. Papamichael and J. C. Hoe, "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs," in *Proceedings of the 2012 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*, pp. 37–46, February 2012.

[18] Y. Huan and A. DeHon, "FPGA optimized packet-switched NoC using split and merge primitives," in *Proceedings of the 2012 International Conference on Field-Programmable Technology*, pp. 47–52, December 2012.

[19] J. H. Bahn and N. Bagherzadeh, "A generic traffic model for on-chip interconnection networks," in *Proceedings of the International Workshop on Network on Chip Architectures (NoCArc)*, 2008.

[20] https://github.com/dsdnu/TreeNoC.