# Managing HBM's bandwidth in Multi-Die FPGAs using Overlay NoCs

by

Srinirdheeshwar Kuttuva Prakash

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

## Author's Declaration

## Abstract

We can improve HBM bandwidth distribution and utilization on a multi-die FPGA like Xilinx Alveo U280 by using Overlay Network-on-Chips (NoCs). HBM in Xilinx Alveo U280 offers 8GBs of memory capacity with a theoretical maximum bandwidth of 460 GBps, but all the thirty-two HBM ports in Xilinx Alveo U280 are exposed to the FPGA fabric in only one die. As a result, processing elements assigned to other dies must use the scarcely available and challenging to use Super Long Lines (SLL) to access the HBM's bandwidth. Furthermore, HBM is fractured internally into thirty-two smaller memories called pseudo channels. They are connected together by a hardened and flawed cross-bar, which enables global accesses from any of the HBM ports, but introduces several throughput bottlenecks, degrading the achievable throughput when the entire memory space is used.

An Overlay Hybrid NoC combining the features of Hoplite and Butterfly Fat Trees (BFT) NoC offers a high-frequency solution for distributing HBM's bandwidth across all three dies, as well as overcoming the throughput bottleneck introduced by the internal cross-bar. The Hybrid NoC combines multiple high-frequency Ring NoCs for inter-die communication and Butterfly Fat tree NoCs for intra-die communication. In addition, the routing capability of the NoC can be modified to supplant the HBM's internal cross-bar for global accesses. We demonstrate this in Xilinx Alveo 280 using synthetic benchmarks and two application-based benchmarks, Dense matrix-matrix multiplication (DMM) and Sparse Matrix-Vector multiplication (SPMV). Our experiments show that NoCs can improve throughput utilization by as much as $\times 8.6$ for single-flit global accesses, $\times 1.7$ for multi-flit global accesses with burst length 16, and as much as $\times 1.4$ for SpMV benchmark.

**Acknowledgements**

I want to thank my supervisors, Professor Nachiket Kapre and Professor Hiren Patel, for their guidance and help throughout my Master's journey. Their expertise, feedback, and perspectives have been the driving force of this work, without which this thesis would not have been possible.

I also want to thank the members of my thesis reading committee, Professor Rodolfo Pellizzoni and Professor Ziqiang Patrick Huang, for their time and effort in reading my thesis.

Furthermore, I want to extend my gratitude to all the members of WatCAG, with a special mention to Ian Lang Elmore and Alireza Mellat. Their inputs and feedbacks have played a pivotal role in the progression of this work.

Being miles away from home and with constant lockdowns, quarantines, and working from home, the past few years have been enormously difficult. However, my beloved friends, Aishwarya Aravamuthan, Alireza Mellat, Valliammai Periakaruppan, and Sham Srinivasan, have been the only semblance of normalcy, and they have helped me withstand these challenging times. I will be forever thankful for their compassion and friendship.

Above all, I want to thank my loving parents, Prakash K.R. and Sabitha K.P., and my dear sister Srinanditha K.P. for their unconditional love, persistent motivation and support throughout my masters. Their constant support has kept me going throughout the masters, helping me push through difficult and stressful periods. I am deeply indebted to them.

## Dedication

To my parents and my sister.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the post-Moore era, FPGAs are becoming essential platforms for hardware acceleration of contemporary workloads such as machine learning [37], graph analytics [5], database management [4], high-speed networking [34], among others. Due to the nature of these applications, access to high bandwidth and large storage is essential for such workloads. However according to Xilinx [45], with four DDR4 running with a bandwidth of 21.3 GB/s, the maximum feasible bandwidth is merely 85.2 GB/s. High Bandwidth Memory (HBM) is a game-changing technology that can provide bandwidth up to 460 GBps, five times higher than DDR4. It is an in-package DRAM memory technology [38] that uses through-silicon vias to interconnect vertical stacks of multiple DRAM chips. For example, the Xilinx Alveo U280 FPGA [46] ships with two HBM stacks totalling 8 GBs capacity with a maximum theoretical throughput of 460 GBps. The high bandwidth comes from its 32 independent channels, exposed to the fabric as 32 AXI ports, each 256 bits wide, operating at 450MHz [48]. When the processing elements are restricted to access only the channel they are connected to; the achievable throughput is close to the theoretical maximum. However, in a realistic setting where the processing elements are allowed to access any channel, the internal crossbar of HBM fails to sustain the high throughput [39][6][40].

Furthermore, modern FPGAs are built from multiple dies connected together using technologies like Xilinx Stacked Silicon Interconnect (SSI). For instance, the Alveo U280 is comprised of three such dies called Super Logic Regions (SLRs)[49] connected by a limited number of Super Long Lines (SLLs)[51]. For example, in Alveo U280, only 23K SLL lines are available per SLR pair. The SLLs are a scarce resource, and they are also challenging to use for nets with feedbacks, such as a Valid-Ready path used in AXIS protocols. Yet, the HBM ports in Alveo U280 are accessible only from SLR0 [46]. Therefore, effectively distributing the bandwidth of HBM across all SLRs is a challenging and crucial task for a chip spanning design. This thesis proposes using Overlay NoCs, such as BFT and Hoplite, to overcome the aforementioned throughput bottleneck and the HBM bandwidth distribution problem. To that end, we adapt the NoCs to support HBM interfacing and customize the routing scheme to route memory transactions to the target channel, eliminating the need for the flawed Xilinx HBM internal crossbar. Similarly, we modify our NoCs to aid in SLR crossing, making it possible to distribute HBM bandwidth across all three SLRs of Xilinx Alveo U280 while also providing PE to PE communication between and within SLRs. We also identify the limitations of Hoplite and BFT NoCs, and to address them, we propose a Hybrid NoC that combines features of Hoplite and BFT.

## 1.1 Objective

In this thesis, we investigate the efficacy of using Overlay Network-On-Chips (NoCs) such as Butterfly Fat Tree and Hoplite NoC to address the problems in using HBM in Alveo U280. First, we propose modifying the routing logic to provide global access by routing the memory packets to their respective *home* AXI ports. This *homing* action of the NoC will completely supplant the need for the HBM's internal crossbar to provide full global access, thereby circumventing the bottleneck affecting the throughput. Secondly, the intrinsic nature of Overlay NoCs enables chip-spanning designs. We exploit this feature of the NoCs to cross SLR boundaries while minimizing SLL resource utilization and maximizing the achievable frequency. We also identify the shortcomings of BFT and Hoplite in meeting our objective, and to resolve them, we propose a new Hybrid NoC that borrows features of BFT and Hoplite. The proposed Hybrid NoC uses Ring NoCs to provide inter-SLR communication and BFTs to provide intra-SLR communication. Finally, we show that Overlay NoCs improve the throughput utilization per port through conducting hardware experiments in Alveo U280 with synthetic and application-based workloads such as Sparse Matrix Vector (SPMV) and Dense Matrix Matrix (DMM). Our experiments show that using an NoC improves throughput utilization by a factor of 8.6 times for single-flit global accesses and 1.7 times for multi-flit global accesses with burst length 16. Whereas, for the SpMV benchmark, the NoC offers a speed-up of up to 1.4 times.

## 1.2 Contributions

The main contributions of our work are:

1. RTL implementation of the BFT, and Hoplite NoC modified for routing memory packets to their corresponding HBM AXI ports. In addition, the routing logic is also augmented to support multi-flit operations to facilitate AXI streaming packets of burst lengths of more than one.
2. Design and implementation of a new Hybrid NoC that combines the features of the BFT and Hoplite NoCs. The proposed NoC borrows vertical rings from the Hoplite to provide inter-SLR communication, and it uses BFT for intra-SLR communication.
3. HLS based application benchmarks such as Dense Matrix Multiplication and Sparse Matrix-Vector multiplication, and a synthesizable RTL-based synthetic benchmarking PE.
4. Evaluation of the performance bottleneck on HBM throughput and the impact of using NoC overlays on HBM throughput utilization in Xilinx Alveo U280 under various synthetic and application-based benchmarks.
5. Multi-SLR spanning implementations of the NoCs for a chip-wide bandwidth distribution.

## 1.3 Thesis Organization

The remainder of the thesis is organized as follows

- **Chapter 2** gives detailed background information on the HBM in Alveo U280, Overlay NoCs, Vitis and Vivado HLS. This chapter also presents a literature review of related works.
- **Chapter 3** presents the main idea of this thesis. It includes the architectural and logical changes made to the reference NoCs to make it HBM compatible. It also identifies the shortcomings of the reference NoCs and proposes a new Hybrid NoC to address these shortcomings.
- **Chapter 4** explains the synthetic and application-based benchmarks used in our experiments.
- **Chapter 5** explains the evaluation and hardware setup used in our experiment, and then the results of the conducted experiments are also presented.
- **Chapter 5** concludes the thesis and outlines possible future research avenues.

# Chapter 2

# Background and Motivation

This chapter first presents the necessary background information on Xilinx HBM in Alveo U280, followed by Overlay NoCs. Then, we review existing literature on related works which involve interfacing processing elements to the HBM. Finally, we briefly explain the tools used in this thesis, such as Vitis and Vivado HLS, and explore the impact of the toolchains on our designs.

## 2.1 Challenges of HBM in Alveo U280

### 2.1.1 Xilinx HBM's Internal crossbar

The HBM in Alveo U280 offers a theoretical maximum throughput of 460 GBps with a memory capacity of 8GBs. This high throughput is made possible by the internal organization of the HBM [48] (Figure 2.1), where the 8GBs of memory space is divided into two stacks. Each stack is further subdivided into 16 Pseudo Channels (PC) and controlled by eight Memory Controllers (MC). So, in total, the HBM is internally divided into 32 pseudo channels, each with a capacity of 256 MBs and a data width of 64 bits. The pseudo channels are exposed to the FPGA fabric as 32 AXI3 ports, and to reduce the clock frequency requirements in the fabric, the ports operate at a 4:1 frequency ratio. Thus, the exposed AXI3 ports are 256 bits wide rather than 64 bits wide. Consequently, the total bandwidth of the HBM is split across the 32 ports, and it is crucial to utilize all the HBM ports to fully exploit the massive throughput offered by HBM. The HBM memory addressing reflects this organization. HBM uses 33 bits addressing, where the most significant 5

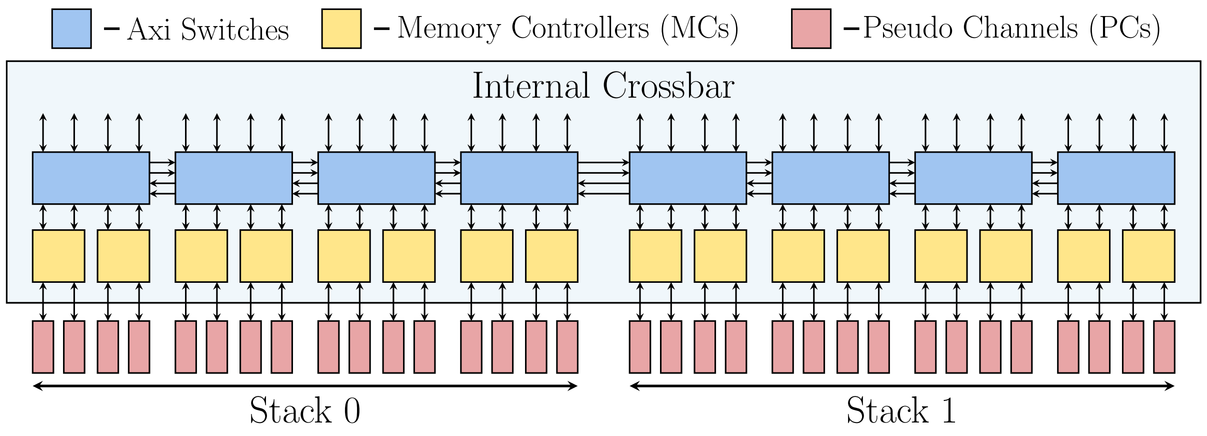


Figure 2.1: Internal Organisation of HBM in Alveo U280

bits, 32 down to 28 points to the *home* pseudo channel and the corresponding *home* AXI port, we call these 5 bits ***port_id***. Bits 27 down to 0 dictates the offset within the pseudo channel. The smallest beat size that the HBM accepts is 256 bits wide; therefore, the 5 least significant bits are unused. The AXI ports exposed to the FPGA fabric operate with AXI clock whose frequency is dictated by the user design, whereas the memory controller operates with a different clock called the HBM clock [48]. The clock domain crossing is handled by the AXI switches present in the internal crossbar.

It is essential to enable access to any pseudo channel from any AXI port. To that end, a hardened internal crossbar, as shown in Figure 2.1, is used. Four pseudo channels, the corresponding two memory controllers, and the four AXI ports are interfaced together using an AXI switch. The AXI ports from the fabric are the masters, whereas the memory controllers and the pseudo channels are the slaves [48]. We call this grouping a ***bank***. Within a bank, the AXI switch acts as a full throughput 4×4 crossbar, where any AXI port can access any pseudo channel without loss in throughput.

However, there are eight such banks in the crossbar, and the communication between them is enabled by four lateral links for each neighbouring pair of banks. The throughput bottleneck of HBM stems from these lateral links. Figure 2.2 offers a closer look at the first two banks and the associated lateral links between them.

To support a full throughput transactions between the Bank 0 and Bank 1, we need four lateral links in each direction. However, only two lateral links are available. As a consequence:

1. Each lateral link is shared by two masters (Fig 2.2), limiting the available throughput

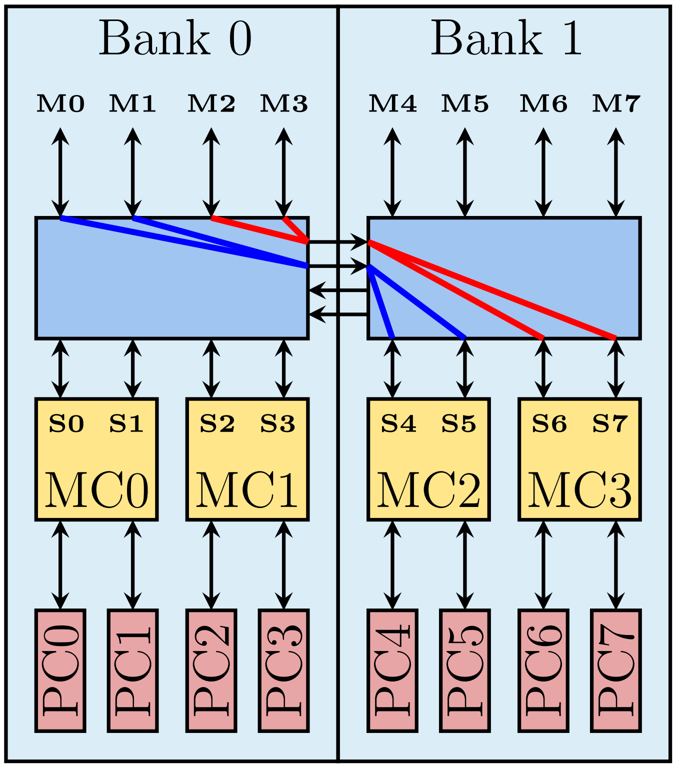
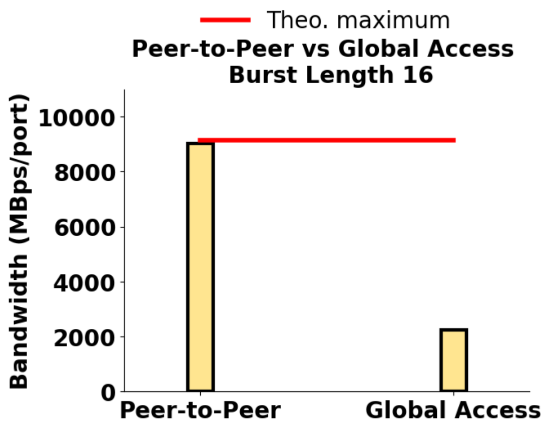


Figure 2.2: Bank 0 and Bank 1. M0 and M1 share the bottom lateral link, whereas M2 and M2 share the top lateral link.

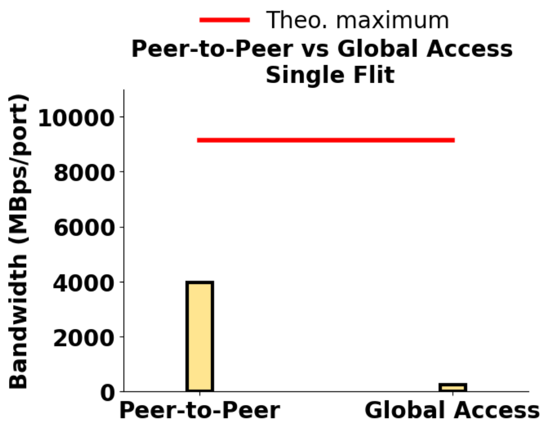
to 50% of the maximum throughput if all masters want to use the lateral links. For instance, with cross-stack accesses (memory accesses that go from one HBM stack to another), the expected performance is a mere 33% of the maximum throughput [47].

2. For lateral links, a dead cycle is required to switch between the masters during writes, which also degrades the achievable throughput, and the extent to which it degrades depends on the burst lengths. Smaller bursts and frequent switching between masters hurt the throughput more than longer bursts, and infrequent switching [48].

The effect of this bottleneck can be seen in Figure 2.3, where the achieved throughput drops by atleast a factor 4, when processing elements make global accesses. When the synthetic benchmarking processing elements, as described in Section 4.1, interfaced to the HBM ports, are restricted to access only the pseudo channel connected to, the achieved throughput is around 98% of the theoretical maximum at 300 MHz. However, when this



(a) Burst Length 16



(b) Single Flit

Figure 2.3: Bar plot comparing the achieved throughput for Peer-to-Peer (P2P) accesses and Global accesses. For P2P accesses with burst length 16, the achieved throughput is 98% of the theoretical maximum. However, when PEs are allowed to make global accesses, the throughput drops by a factor of 4 . Whereas, for single flit P2P accesses, the achieved throughput is merely 43%. When global accesses are allowed this drops by a factor of 10.

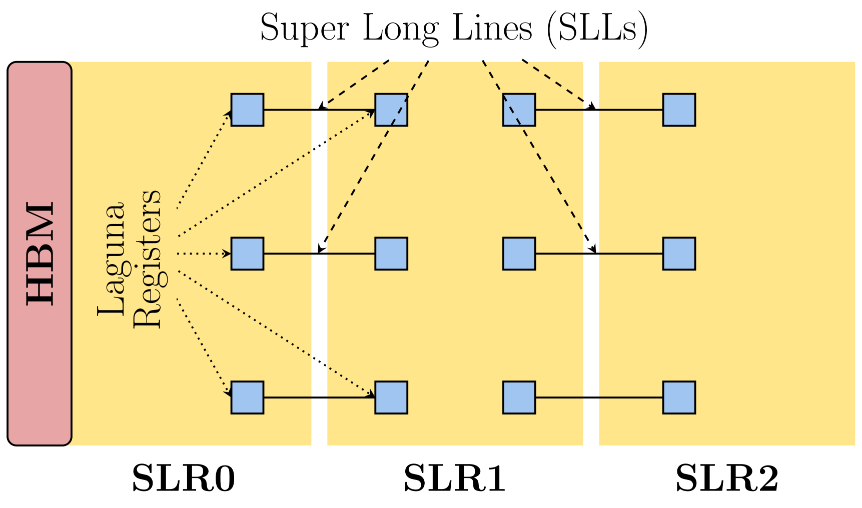


Figure 2.4: Super logic regions in Alveo U280

restriction is lifted, and the PEs can access any pseudo channel, the throughput drops by four times.

To conclude, the throughput bottleneck of the HBM stems from the lateral links that connect banks together, which also is essential in providing global access from any port. Therefore, the achievable throughput drops when the hardened internal crossbar is relied upon to support full global access from any of its AXI ports. Instead, if the memory transactions do not cross the bank boundary, the lateral links are not required, and hence the throughput performance remains unaffected.

### 2.1.2 Super Logic Regions

As shown in Figure 2.4, Xilinx Alveo U280 is comprised of three stacks of dies called SLR0, SLR1, and SLR2. The connection between the SLRs is established using Super Long Lines(SLLs) [51] that connect Laguna registers on either side of the SLRs. As explained in Section 2.1.1, the 32 ports of the HBM are exposed to the fabric as 32 AXI3 ports, each with a data width of 256 bits. However, the ports are not uniformly distributed across the SLRs. All of the 32 ports are available to interface only in SLR0 [46]. So, in a design spanning multiple SLRs, processing elements placed in SLR1 and SLR2 must use the limited and challenging to use SLLs to access SLR0 and, by extension, HBM ports. For instance, in Alveo U280, only 23,000 SLLs are available between each SLR pair.

SLLs are challenging to use because they incur a considerable path delay if the SLR crossing net is not between two Laguna registers [51], lowering the attainable frequency. This is achievable through careful pipelining and floorplan constraints in a strictly feed-forward design. Whereas making this crossing can be challenging for designs with handshake protocols, such as AXI Streaming. Adding simple pipeline stages would break the functional correctness, and the use of shadow registers to ensure it would require the inference of LUTs after the registers. Therefore, an SLL crossing between two pipeline stages comprising shadow registers would not be between two laguna registers. Instead, it will be between a LUT and FF, making the SLL crossing expensive in terms of time delay, as shown in Figure 3.4. Consequently, moving processing elements that interface with HBM ports from SLR0 to other SLRs, as required to fully utilize the available resources in the FPGA chip, can be a challenging and tedious endeavour.

To summarise, the problems with Alveo U280 are two-fold. 1) The internal crossbar of HBM introduces bottlenecks that degrade the throughput performance when full global accesses are made. 2) The placement of HBM ports in SLR0 makes it challenging to distribute HBM's bandwidth to SLR1 and SLR2.

## 2.2 Overlay Network-on-Chips

FPGA overlays NoCs are a scalable and configurable communication fabric for connecting multiple subsystems or IPs in an FPGA or SoC [19]. A direct or peer-to-peer communication network between processing elements in an FPGA quickly becomes infeasible with the increase in the number of processing elements. Overlay NoCs offers a scalable and cost-effective solution to connect multiple processing elements in an FPGA. Furthermore, it enables the processing elements to be tessellated across the chip, leading to better utilization of the available FPGA resources.

A topology is a structure or the arrangement of the switches and their connections within a network. In Overlay NoCs, the deterministic nature of the underlying hardware and its applications allows for a regular and simple topology. Some of the popular Overlay NoC topologies are mesh [22][35] (Fig 2.5), torus [25][11], and fat-trees [24][31]. A routing algorithm is required to guide the packet injected into the network to the desired destination, where at every switch, the routing algorithm is applied to the incoming packets to determine the desired output ports. Broadly, the NoCs can be classified into two groups - Single-Flit and Multi-Flit networks.

In the following sections, we briefly explain the differences between single-flit and multi-flit networks and explain the switching strategies that are used in multi-flit networks. Then, we explain the concept of deadlocks and a possible solution to avoid them - Virtual channels. With the basic terminologies and concepts explained above, we then introduce two topologies, BFT and Hoplite, used in our designs.

### 2.2.1 Single Flit vs Multi Flit networks

A flit can be defined as the smallest unit of information that the NoC routes, and it is determined by the bit-width of the NoC links. A message or a packet larger than the flit size of the network is broken down into several flits, which are then routed by the network [30]. In a single-flit NoC, every flit must bear the addressing information, such as the destination and source address. As a result, flits of the same message can be routed independently, reducing design complexity at the switch level. However, since every packet carries the routing information along with the data, single-flit networks have a larger overhead in terms of bit width than multi-flit networks. Furthermore, for burst operations such as AMBA AXI protocols, where the sense of stream must be preserved, single-flit routing could lead to interference from other streams, requiring complex designs at the receiver to

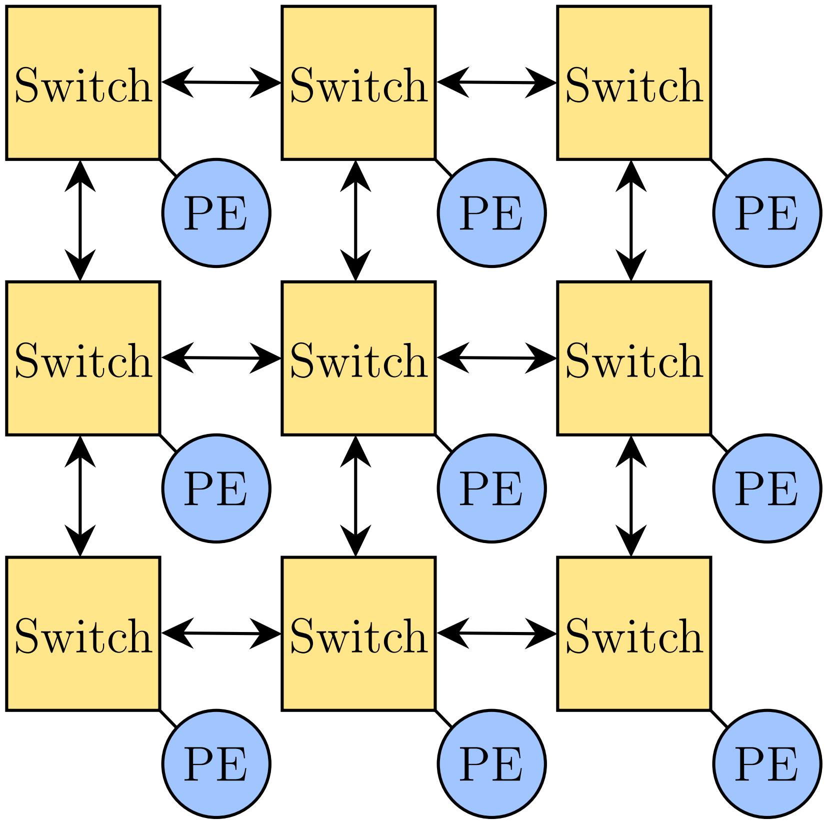


Figure 2.5: Example of a Mesh NoC

handle interference from several streams. Therefore, single-flit networks are not ideal for supporting burst operations.

In a multi-flit NoC, only the header packet, i.e. the first packet, carries the addressing information. In this type of network, the switches must retain the routing choice until the entire stream is routed through it. In contrast to single-flit routing, in multi-flit routing, the flits of the same message are routed together as a stream, making it ideal for burst operations. There are two possible switching strategies for a multi-flit network - circuit switching and packet switching [36].

### Circuit Switching

In circuit switching, the path between the source and the destination is first established, and then the data flits are transmitted through the network. First, a header packet with address information is injected into the network. Then, the desired path is reserved at each switch, and the header packet is moved along the requested path. When the header packet reaches the destination, an acknowledgement is sent back to the source, finally prompting it to transmit its data flits [10]. The path reservation is released when the transmission is done. In a network with only one physical channel and no virtual channels, header and data flits of other streams cannot utilize the reserved path until the reservation is lifted. Thus, circuit switching is ideal for infrequent and long messages. However, for frequent and short messages, the time taken to establish the path between the source and destination might be longer than transmitting the message as a whole together with the header packet [36].

**Packet Switching**

In packet switching, the entire message (header and data flits) are routed together, where the path is allocated and deallocated as the message travels through the network. There are two major approaches to implement this. In the store and forward approach, [36][10], the entire message is stored at each switch, and then the message is forwarded to the next switch as a whole. This method requires buffers at every port in every switch, leading to higher resource utilization. The other approach is wormhole switching [36]. In this approach, the flits are injected into the network in a pipelined fashion. Switches use the header flit to reserve the desired output for the header (and by extension the stream), and the header flit is forwarded to the desired port. Since the header flits are the only flits bearing the addressing information, the routing choice made by the switch for the header packet is retained until the last flit passes through the switch. Consequently, the entire message need not be buffered at every port in every switch, leading to lower resource utilization. However, if the header flit is back pressured due to contention, the flits following it and other contending messages are back pressured as well. Once the contention is resolved, the flits start progressing again. Therefore, packet switching is preferred for frequent and shorter messages, leading to high link utilization [27].

### 2.2.2 Deadlock

In a packet-switched NoC without Virtual Channels and one physical channel, there might arise a situation where the packets form a series of dependencies that prevents them from progressing, leading deadlock. In Figure 2.6, we depict an example of deadlock, where is Packet A is blocked by Packet B, Packet B is blocked by Packet C, Packet C is blocked by Packet D, which is in turn blocked by Packet A, creating a circular dependency and causing the network to get deadlocked. For a deadlock-free operation of an NoC, steps must be taken to avoid or recover from a deadlock [9][17]. For instance, deadlock can be avoided in a mesh network by imposing restrictions on certain turns. An example of such an algorithm is Dimension ordered routing, also known as XY routing [9]. Another method to avoid deadlock is through the use of Virtual Channels [17].

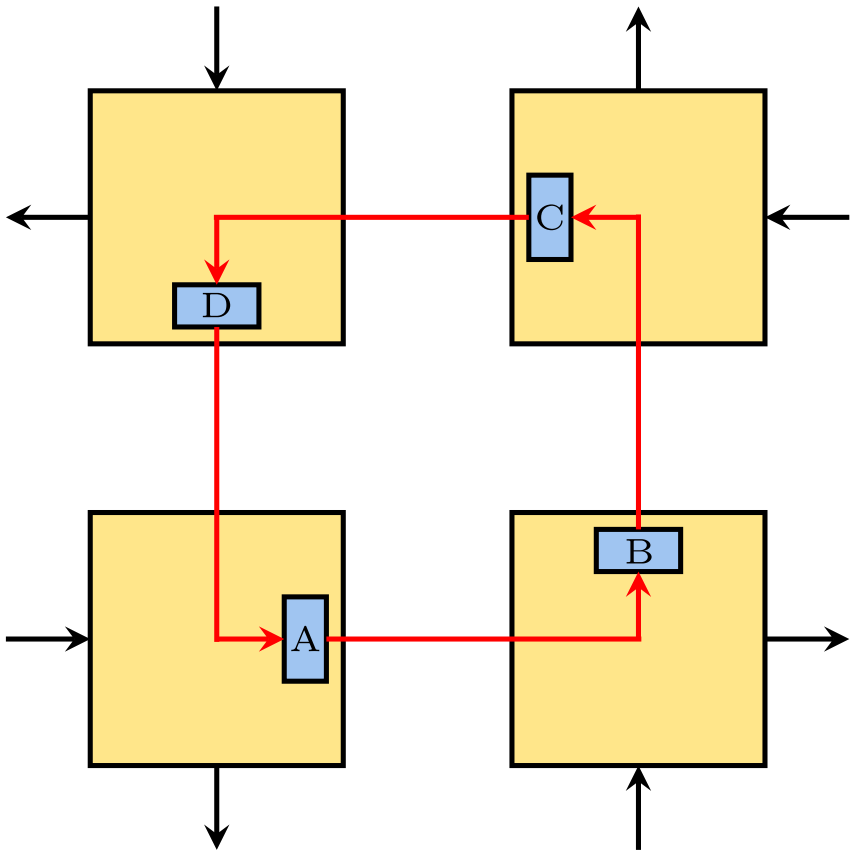


Figure 2.6: Example of a deadlock

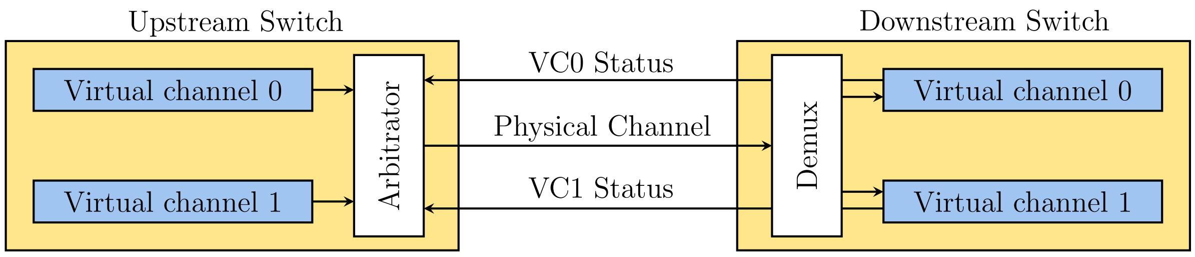


Figure 2.7: example of a Virtual Channel implementation

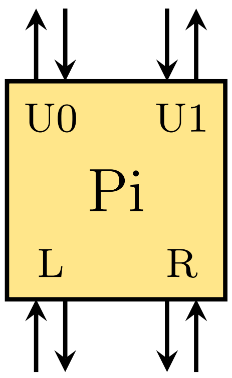
### 2.2.3 Virtual Channels

Virtual channels separate a physical channel into several logical channels, as shown in Figure 2.7. They are managed by buffers at either end of the physical channel, and typically, the virtual channels share the physical channel in a time-multiplexed fashion, where only one packet from any virtual channel is allowed to enter the physical channel in a cycle [27]. For a flit to proceed, the status of the virtual channels in the downstream switch must be known before it is launched into the physical channel. Therefore, there needs to be a buffer management system in place to communicate the status of virtual downstream channels to the upstream switch. A credit-based system is one such buffer management system [10], where the upstream switch keeps track of the number of free slots in the downstream virtual channel buffers. The downstream module then notifies the upstream switch whenever a slot is freed. We use only two virtual channels in our switches. Therefore we take a simpler approach where we use separate signals (one for each virtual channel) to indicate the status of the downstream buffers to the upstream switch, as indicated in Figure 2.7.

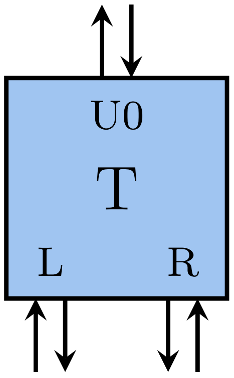
Virtual channels have several advantages. First, as mentioned above, they can be used to avoid deadlock in multi-flit networks [17]. For instance, by assigning some of the packets to different virtual channels, in the example given in Figure 2.6, the circular dependencies can be broken, allowing the packets to move ahead. Furthermore, they also improve the utilization of the physical channel by allowing packets to be routed around other blocked packets by simply switching them over to a different virtual channel [53].

### 2.2.4 Butterfly Fat Tree (BFT)

BFT is a hierarchical tree-based NoC topology, where the PEs are interfaced at the leaf nodes, and the switches are the internal nodes of a fat tree. This thesis uses a 2-ary BFT



(a) Pi switch



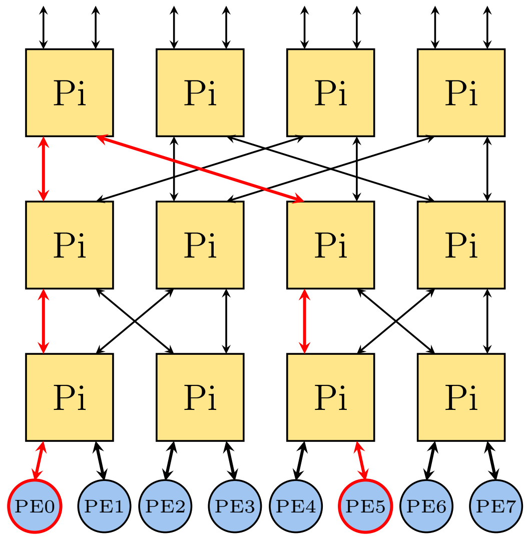
(b) T switch

Figure 2.8: Switches used in BFT NoC

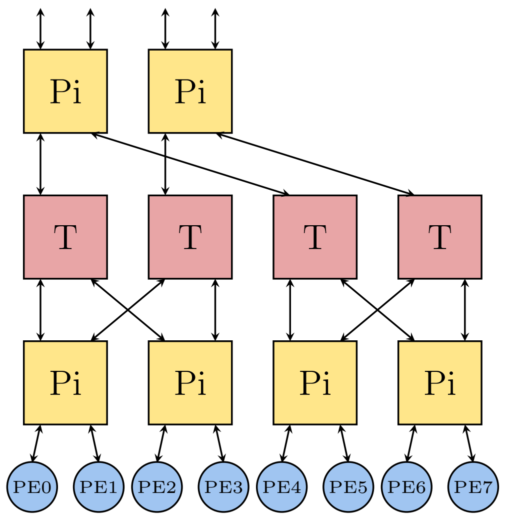
NoC with lightweight flow control, as described by Malik et al. [32], as our reference design. The building blocks of BFT NoC are T and Pi switches. A Pi switch has a total of eight uni-directional AXIS ports as shown in (2.8a). This architecture allows for a maximum throughput of four packets per clock cycle when there are no contention, i.e L → U0, R → U1, U0 → L, and U1 → R. In contrast, a T switch has six ports, as shown in (2.8b), limiting the throughput to two packets per clock cycle when there are no contention, i.e L → U0 or R → U1, and U0 → L or U0 → R.

For any configuration, only one type of switch can be used at a given level, and by varying the type of switch used at each level, the network bisection bandwidth is customized to meet the application requirements, where the bisection bandwidth is defined as the bandwidth available between two partitions of a network. For example, XBAR, as shown in 2.9a is a BFT configuration constructed using only Pi switches at every level. This configuration has a bisection bandwidth of $O(N)$ and offers the maximum bandwidth out of all possible configurations. On the other end of the spectrum, the TREE configuration is constructed using only the T switches at every level and has a bisection bandwidth of $O(1)$. MESH0 and MESH1 (Fig.2.9b) are the other two possible configurations, and they are constructed by following pi-pi-t-t, and pi-t-pi-t patterns, respectively. The BFT NoC used as reference is an ideal network for interfacing PEs with the HBM due to its Deterministic routing, flow control, in-order delivery, and livelock-free routing [32]. However, as far as to our knowledge, BFTs have not been used to interface processing elements to the HBM.

Packets injected by the processing elements contain the destination address, source address, and payload data. Once injected, the packets first ascend the network until it reaches a common ancestor between the source and the destination. In this version of the



(a) BFT in XBAR configuration. The path highlighted in red is the path taken by the packets injected by PE0 with the destination set to PE5.



(b) BFT in MESH1 configuration.

Figure 2.9: BFT in XBAR and MESH1 configurations.

BFT NoC, the uphill freedom in Pi switches is limited. Packets that arrive at the L port can ascend only through U0. Similarly, packets that arrive at port R can ascend only through U1. When the packet reaches the level of the common ancestor indicated by the bit position of the common prefix of both addresses, it turns (L$\rightarrow$ R, R$\rightarrow$ L) and starts to descend towards the destination. During the descent, no downhill freedom is available. The choice of the downhill port at level l is dictated by bit l of this destination address. If it is set, the R downhill port is chosen. Otherwise, L is chosen; This setup leads to contention between turning packets and descending packets. A round-robin arbitration is used to determine the winner [32]. The winner is routed to the requested port and subsequently the next level. The loser is stored in a shadow register, and the losing port gets back-pressured. The presence of shadow register and back-pressure dictates a requirement for a handshake-driven protocol at each port. For compatibility with Xilinx IPs, the AXI-streaming protocol is used. Figure 2.9a shows the deterministic path taken by a packet injected by PE0 with the PE5 as the destination.

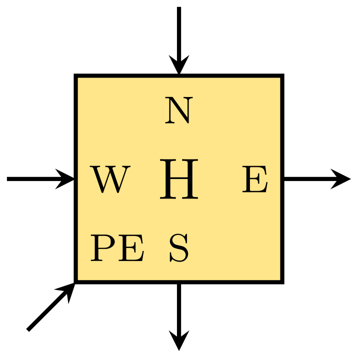


Figure 2.10: Switch used in Hoplite NoC

### 2.2.5 Hoplite

Hoplite NoC, as proposed by Kapre et al. [26] is the reference design for the Hoplite network used in this thesis. It is a uni-directional 2D torus NoC topology where the network switches are arranged in a grid-like fashion determined by its $R \times C$ dimensions, where $R$ is the number of rows and $C$ is the number of columns in the NoC. For instance, in Figure 2.11 we have depicted an example Hoplite NoC with 9 PEs arranged in 3×3 dimension, where we have three rows and three columns of PEs interfaced to the Hoplite switches.

The building block of a Hoplite network is the Hoplite switch shown in 2.10. The switch has three input ports, North, West, and PE, and two output ports, South and East. Unlike the BFT NoC, Hoplite does not use any queuing of packets. Instead, it uses deflection-based routing with a turn restriction policy called dimension-ordered routing (DOR). In this policy, the packets first route along the horizontal link (W$\rightarrow$ E), and at the appropriate column, the packet turns and routes along the vertical link (N$\rightarrow$ S). DOR reduces the resource costs and also ensures deadlock-free operation for single flit packets. However, due to the use of deflection-based routing, the in-order delivery guarantee is compromised [26].

Rather than a round-robin arbitration, the contention of ports is handled by a priority order. When the south port is contended by packets at North, West and PE ports, the packet at the North port is given priority, and it moves to the requested south port. The packet at the West port is deflected to the East port, and the packet at the PE port is rejected. To prevent the loss of data, it is the responsibility of the PE to hold on to the rejected packet. Due to DOR, only packets at the West port and PE port can contend the East port. In this case, packets that arrive at the West ports are given the higher priority,

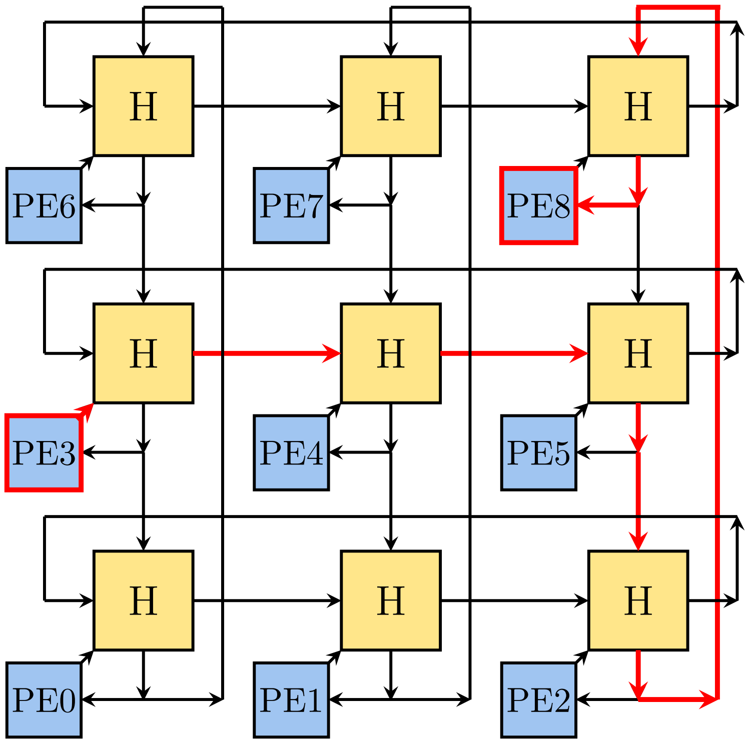


Figure 2.11: 3×3 Hoplite NoC. The path highlighted in red is the path taken by the packets injected by PE3 with the destination set to PE8. Note that the south port is begin multiplexed as the input port to PEs as well.

moving on to the East Port. Similar to the contention of South port, the losing PE packets are rejected. In Figure 2.11 we highlight the path taken by packets travelling from PE3 to PE5. The packets first travel in the horizontal direction, and when it reaches the column PE5 is connected to, the packets turn and travel around the vertical direction until they reach PE5. One quirk of the underlying Hoplite is the multiplexed use of the south port. Apart from being used as the south port for packets travelling along the vertical link, it also acts as the input port to the PE. In the example, we have used a single channel Hoplite NoC, where each PE can ingress and egress packets from only one channel.

## 2.3 Related Works

### 2.3.1 HBMCONNECT

HBMCONNECT, proposed by Choi et al. (2020) [6], is an HLS based FPGA interconnect designed for interfacing processing elements to the HBM. The interconnect aims to eliminate the dependence on the HBM's internal crossbar, thereby circumventing the throughput bottleneck. The interconnect is constructed using several 2x2 switching elements called Mux-Demux switch connected together in a multi-level butterfly topology. The memory packets are then routed to their corresponding AXI ports by the Mux-Demux switches. When there is no contention, the Mux-Demux switch can route 2 data packets per cycle, but on average, the switch outputs 1.5 data per cycle. At every level, the uphill output port is chosen based on the memory address of the input packets. Therefore, a network with 16 processing elements interfaced to 16 HBM ports, requires four stages of Mux-Demux switches to route the AXI memory transactions to the target HBM AXI port, eliminating traffic on the lateral links of the HBM's internal crossbar. The authors perform two case studies using Bucket Sorting and Merge Sorting PEs, and their experiments show that HBMCONNECT improves the $BW^2/LUT$ metric by up to 211 times and the $BW^2/BRAM$ metric by up to 85 times.

#### Problems with HBMCONNECT

1. The proposed interconnect does not support PE to PE communication and supports only PE to memory communication. To support PE to PE communication, either an NoC needs to be instantiated, or the memory must be used to exchange information.
2. The use of full AXI protocol in their switches makes SLR crossing a challenge. As a result, the authors of the paper floorplan their design to utilize only SLR0, leaving SLR1 and SLR2 unused.

### 2.3.2 2GRVI Phalanx

Jan Gray (2018) [20] has previously used a single channel Hoplite NoC architecture with a dimension 15x15 to interface 222 SC-V RV64I Processor Cluster Arrays with HBM. The 15 vertical links of the network are interfaced to 30 HBM ports via AXI-switch-MC-HBM2 bridges, and with an operating frequency of 300MHz, the NoC can support bi-section bandwidth of 2.3Tbps. However, this structure effectively halves the maximum theoretical throughput that can be extracted from the HBM. Furthermore, while this network can be used to implement a chip-spanning design that fully utilizes the resources on offer, the NoC used in their design does not route the memory packets to the corresponding HBM port. Instead, it relies on the HBM's internal crossbar to provide global access, which, as explained before, suffers from throughput bottlenecks.

## 2.4 Tool-Chains

### 2.4.1 Vitis

Vitis unified platform is the merger of several Xilinx tools such as Xilinx SDK, SDSoC, and SDAccel [16]. Vitis is used in two different flows. The first flow, called the embedded development flow, is used as a replacement for Xilinx SDK in developing software for embedded platforms such as Xilinx ZCU102. In the second flow, called the application acceleration flow, Vitis is used as an FPGA-based software acceleration tool for application platforms such as Alveo u280. This thesis primarily uses the Vitis in application acceleration flow targeting Alveo u280 with RTL and HLS kernels.

A Vitis project in application acceleration flow has two components - a host application and a Hardware kernel. The host application, written in OpenCL, is executed in the host system such as an Intel x86 machine, and it uses Xilinx Run Time (XRT) API calls to control the execution of the hardware kernel (written in RTL or HLS C) in the FPGA fabric [50]. It also manages the data flow to and from the FPGA board interfaced to the host system, as shown in Figure 2.12. For instance, a typical host code would first transfer the required contents from host memory to the global memory available in the FPGA board and start the execution of the hardware kernel. Once the kernel indicates the end of the computation, the host code then reads back the results from the global memory and places them in the host memory. For a data center card like Alveo U280, the host code uses a PCI-E interface to communicate with the FPGA board. Therefore, any design that uses Vitis for execution must instantiate and manage a PCI-E IP and other auxiliary modules required to enable the communication between the host and the board.

The PCI-E module is mandatory for any design to execute on board, and therefore, certain FPGA resources are reserved for PCI-E and other auxiliary modules. The chip area reserved for PCI-E is called a static region, and Xilinx advises against using the static region for user design. Furthermore, the hardware kernel and the Vitis modules that change with the kernel are instantiated in the rest of the chip, called the dynamic region.

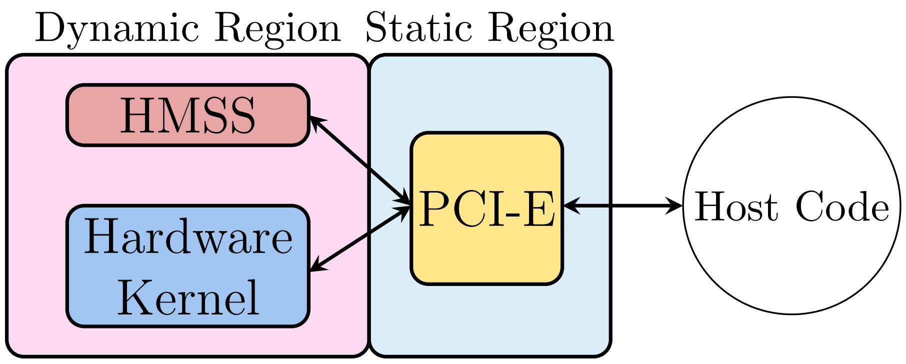


Figure 2.12: Vitis control and data flow.

Consequently, the Vitis consumes some of the FPGA resources and the SLL lines. In Alveo U280, the last CR column is wholly assigned to the static region, as shown in Figure 2.13. Apart from the 2880 SLLs in the static region, the Vitis also consumes 4000 SLLs in the dynamic region. This has significant consequences on our design and floorplan, as will be described in Chapter 3.

### 2.4.2 Vivado High-Level Synthesis

High-level synthesis (HLS) such as Vivado HLS converts higher-level languages such as C/C++ into HDL languages like Verilog or VHDL [43]. The familiarity and simplicity of higher languages enable users to utilize HLS to prototype and deploy large and complex designs quickly. Consequently, HLS tools have been increasing in popularity in recent years. HLS tools like Vivado HLS first construct a control flow graph and a data flow graph from the given program. Then, using these graphs, the RTL code is generated [8].

Without any optimizations, the Quality of Results of the generated RTL hardware is poor. The QoR can be improved by rewriting the higher-level code to reflect the underlying hardware better and using optimization techniques. Furthermore, the protocols used for the interfaces need to be communicated to the compiler. To do this, we need a mechanism to communicate to the HLS compiler about the desired optimizations and to indicate the desired interface protocols. In Vivado HLS, both of these can be done through the means of special macros called pragmas [43]. Among other things, pragmas can be used to reduce latency, improve throughput, direct the compiler to use specific resources, optimize for the area, and control the interface protocols used for the IPs. These pragmas can have profound effects on the QoR of the generated hardware, and in the following, we explain

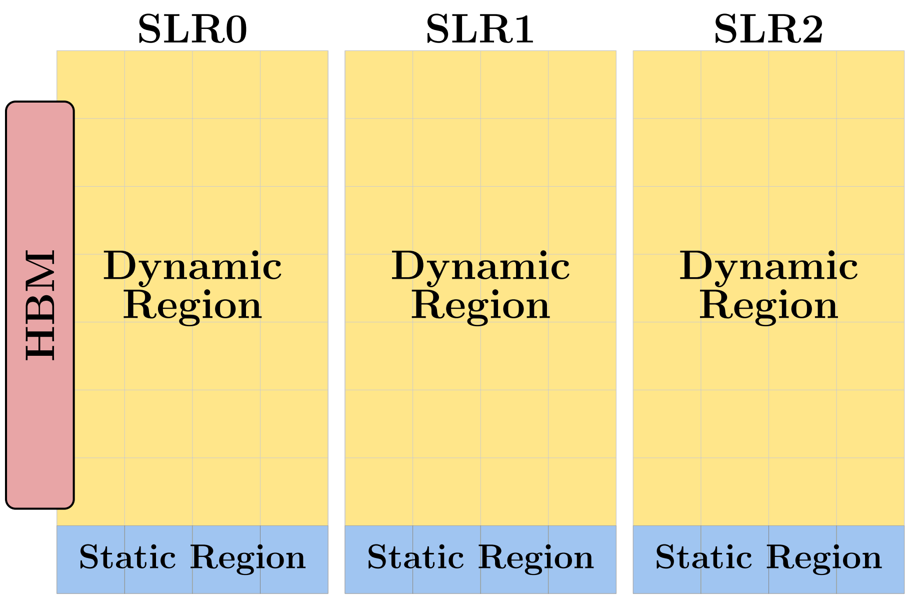


Figure 2.13: Dynamic and Static region in Alveo U280. There are 8 rows and 12 columns of CR in Alveo U280. The first 11 CR columns (coloured in yellow) are assigned to the dynamic region, and the last CR column (coloured in blue) is assigned to the static region.

some of the pragmas that we have extensively used in our designs.

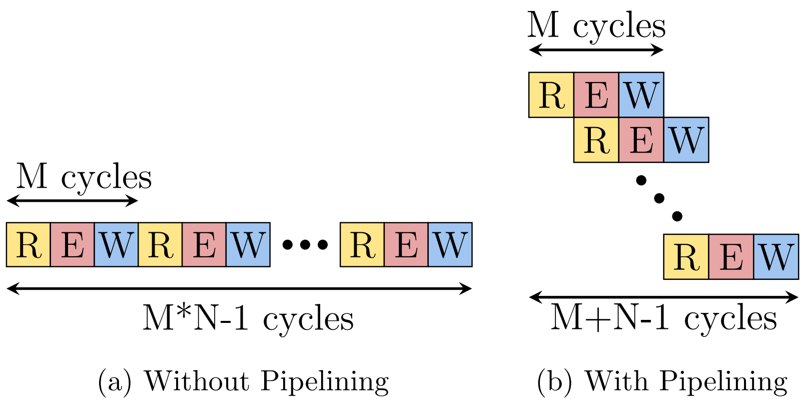


Figure 2.14: Loop Pipeline Directive

### Pipelining

The throughput of the function or loop can be improved by adding pipeline pragma with the initiation interval set to 1. For example, let assume that a function performs three tasks Read, Execute, and Write. Without any pipeline directive, the HLS compiler generates an RTL that executes the different iterations of the loop in sequence, as shown in Figure 2.14a. With the pipeline directive, the HLS compiler schedules the next iteration of the loop immediately at the next cycle as depicted in Figure 2.14b. If the tasks take M cycles to complete, and if the loop iterates for N times. The total latency of the loop without the directive will be $M * N$ cycles, and with the pipeline directive, the total latency will drop to $M + (N - 1)$. Therefore, with the pipeline directive, the total latency of the loop decreases, and new input is accepted at every cycle, making the throughput 1. However, pipeline directive (with initiation interval 1) will fail when applied to loops with inter-loop or intra-loop dependencies. The program code must be rewritten to eliminate the loop dependencies to pipeline the loop.

### Unrolling

A single module is instantiated and pipelined with the pipeline directive, leading to a throughput of 1. Spatial parallelism of the hardware can be exploited by using the unroll directive. The unroll directive with factor set to N creates N independent module which executes in parallel as shown in the Figure 2.15. The unroll directive significantly improves the throughput and increases resource utilization. In the example given in Figure 2.15, the throughput jumps from 1 (for pipelined case) to N when the unroll directive is used. Similar to pipeline directives, care must be taken to avoid inter-loop dependencies.

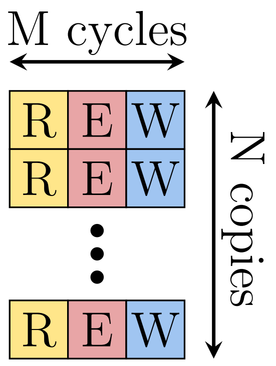
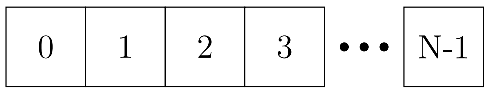


Figure 2.15: Unroll Directive

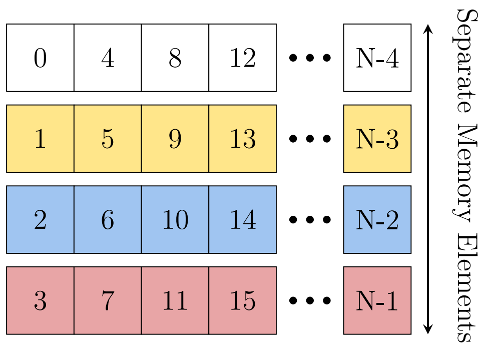
## Array partitioning

Apart from loop dependencies, accesses to the arrays(mapped to memories) prevent complete unrolling or pipelining. Array partition pragma partitions a larger array into multiple smaller memory elements, increasing the amount of data that can be read and written from and to the array. In conjunction with the pipeline pragma or the unroll pragma, array partitioning pragma can effectively improve the latency and throughput of the design. However, it also requires more registers, BRAM and DRAM blocks to implement. There are three different ways to partition the array. They are:

1. **Cyclic**: The smaller arrays are formed by interleaving the elements from the original array. If the defined factor is N, the first N elements are put in separate memory elements. Then the N+1st element is put in the first memory element again. This is repeated until all elements are partitioned. This type of partitioning is ideal when the loop access consecutive elements from the array. An example of this type of partitioning is shown in Figure 2.16, where the factor for partition is set to 4.
2. **Block**: The smaller arrays are formed by splitting the array into continuous and consecutive block defined by the factor N. This type of partitioning is suitable when the loops access are N elements apart.
3. **Complete**: The original array is completely broken down in individual registers. This is suitable when the access patter is random.



(a) Before cyclic partitioning



(b) After cyclic partitioning

Figure 2.16: Cyclic Partitioning

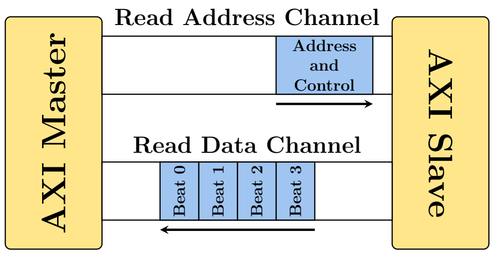
Apart from the loop optimization, the pragmas are also used to control the interface protocols used in the IP. The choice of interface protocols plays a significant role in the hardware operation and the nature of the communication between the IP and the other modules (be it a processor or a memory). We cover some of the interface protocols used in our design in the following section.

### AXI4 protocol

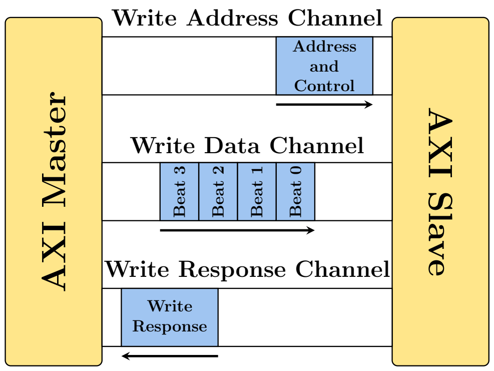
AXI4 is a memory-mapped interface protocol, part of AXI ARM AMBA (Advanced Microcontroller Bus Architecture) specifications. In AXI4, the communication between the master and slave takes place over five different channels, Read Address, Read Response, Write Address, Write data, and Write response channel, using a ready-valid handshake for each channel. The memory transaction, be it read or write, are always initiated by the master, and the data can move in either direction in parallel [44]. One of the essential features of the AXI4 protocol is the burst operation mode. Wherein, the master can write or read up to 256 beats per transaction as shown in Figure 2.17. For example, in a typical write transaction, the master would initiate it by transmitting the address and other control signals, followed by the necessary data beats. Finally, the outcome of the write, success or failure, is communicated to the master by the slave (Figure 2.17b). Similarly, for read operation, the master first transmits the address and control information, after which the slave responds with the requested data in the read response channel (Figure 2.17a). The burst mode of operation significantly improves the throughput for large read and write transactions, and therefore the AXI4 is a popular interface for large input and output arrays with regular access patterns.

### AXI streaming protocol

Like AXI4 protocol, AXI streaming protocol (also known as AXIS protocol) is part of ARM AMBA specifications. However, unlike the AXI4 protocol, AXIS offers only one uni-directional channel for data transmission between the master and the slave, where the data always flows from the master to the slave. In its rudimentary implementation, only three signals are required. A data bus, a single bit ready, and a valid signal [44]. The data is said to be transmitted from master to slave only when the valid signal of the master and the ready signal of the slave is high together in a given cycle. If the data at master is valid, and if the slave is not ready to accept it, the onus is put on the master to hold data until the slave is ready to accept it. One key advantage of using AXIS protocol for the interfaces is the ability to daisy-chain multiple IPs together in series. This property makes it ideal for streaming applications such as communication, Video and Image processing.



(a) Read Operation in AXI4



(b) Write Operation in AXI4

Figure 2.17: AXI4 write and read operation

# Chapter 3

# Main Idea

There are two key shortcomings of HBM in Xilinx Alveo U280. (i) As explained in Section 2.1, HBM in Xilinx Alveo uses an internal crossbar to support global addressing from any AXI port. However, due to the limited lateral links, the throughput utilization degrades significantly for global addressing. This limitation of the HBM can be overcome by reducing or eliminating the traffic in the lateral links. (ii) Furthermore, in Alveo U280, the HBM ports are available only in SLR0, requiring the use of the scarce and challenging to use SLLs to distribute the HBM bandwidth to other SLRs, making effective use of FPGA resources in SLR1 and SLR2 challenging. This thesis proposes using NoC Overlays to address both of these shortcomings of HBM in Xilinx Alveo U280.

In this chapter, we present the necessary architectural and logical modifications to the BFT and Hoplite NoC overlays that enables:

**1. PE-PE and PE-HBM communication**

Processing Elements with AXIS-Compatible data paths to be interfaced with HBM ports while retaining the PE to PE communication property of the underlying NoC.

**2. Homing**

Memory packets are to be routed directly to the corresponding AXI port over the NoC, completely eliminating the traffic in the lateral links of the HBM internal crossbar.

**3. Distribution of HBM bandwidth across SLRs**

Modules in SLR1 and SLR2 in Alveo U280 to access HBM bandwidth by using SLL crossing through the NoC links while minimizing path delay.

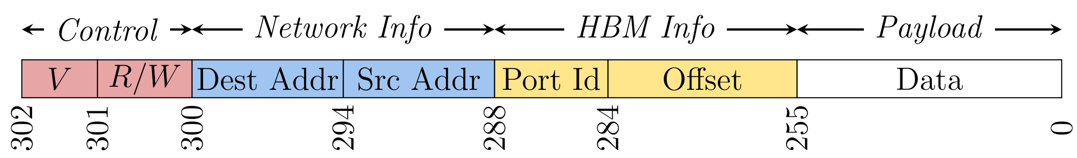
We also examine the advantages and the shortcomings of Hoplite and BFT NoCs used in our designs, and then, based on our observations, we propose a new Hybrid architecture that aims to rectify the flaws of the Hoplite and BFT NoCs.

First, we present the design components that are agnostic to NoC topologies, such as flit design and the HBM-NoC interface. Then, we explain in detail the necessary changes to Hoplite and BFT NoC that make ***homing*** and SLR crossings possible. Finally, we propose the new Hybrid NoC architecture to address the shortcomings of the BFT and Hoplite NoC.

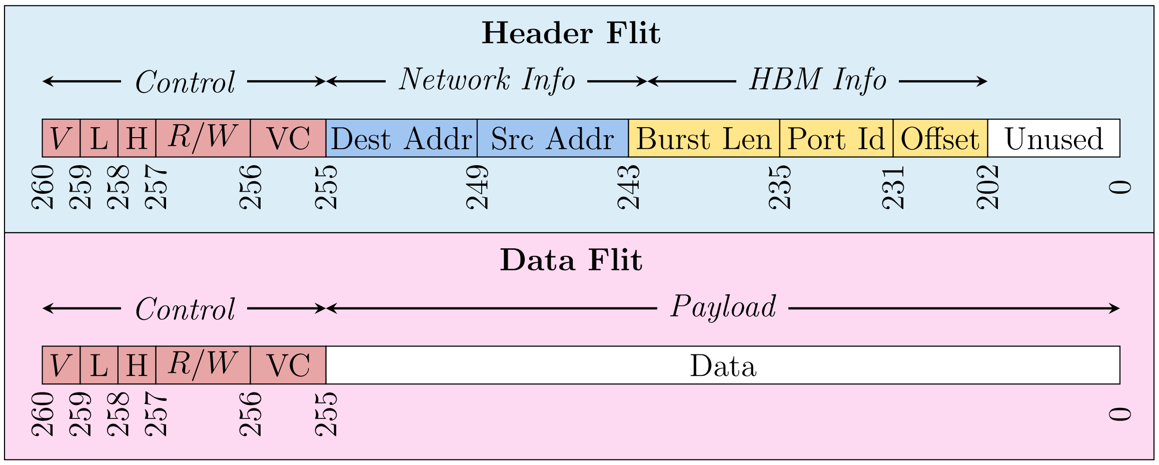
## 3.1 Flit Design

A flit is the smallest unit of information that is routed by the network. In a single-flit network, every flit carries the data and addressing information. Since the objective of our design is to interface PEs to HBM, the single-flit networks used in our design also carry the memory address and additional control bits. The single-flit format used in our design is shown in Figure 3.1a. We set the payload width of our network to 289 bits, enough to carry 256 bits of data to and from HBM and the 33 bits HBM's memory address. The HBM memory address can be split into two sections. The most significant 5 bits are for **port_id**, which is used to determine the pseudo channel and, by extension, the HBM AXI port. The rest 28 bits are for determining the offset within the pseudo channel. Along with the data bits, each flit will also carry network address information such as the source address and destination address, each 6 bits wide (enough to support a network with 64 PEs). Similarly, two control bits, one Valid bit (V) to determine the validity of the packet, and one read-write (R/W) bit, are also used. The Read-Write ($R/W$) bit is used to distinguish between read and write memory transactions. In total, for single-flit operations, the flit width of our NoC is set as 303 bits.

For multi-flit networks, the header packet alone carries the address information, thereby reducing the flit bit-width. In Figure 3.1b we depict the format used for the header flits in our multi-flit network. Similar to the single-flit format, 33 bits are allocated for the memory address and 12 bits for encoding destination and source addresses. However, unlike the single-flit network, in multi-flit networks, the burst length of the stream can be more than 1. Accordingly, we allocate 8 bits to encode the burst length. Furthermore, 5 control bits are used - Valid (V), Last (L), Head (H), Read-Write ($R/W$), and Virtual Channel (VC) bit. The Head bit indicates the start of the stream, and the Last bit indicates the last flit in a stream. The Virtual Channel bit is only used in the Hybrid NoC, and it determines the virtual channel assigned to the stream.



(a) Flit format used in Single-flit Network



(b) Flit format used in Multi-Flit Network

Figure 3.1: Flit format used in our NoCs

For Data Flits, only the control bits (260 down to 256) are retained from the Header flit, whereas the least significant 256 bits (255 down to 0) carry the data. Since the read requests do not require any data flits, they just use the header flits. In comparison, write transactions need to have at least one data flit.

## 3.2 Multi-Flit support

From Figure 2.3, it can be observed that the with burst length 16, processing elements connected to the HBM ports are able to achieve 98% of the theoretical maximum throughput. Whereas with single flit accesses (burst length 1), the achieved throughput is merely 43% of the theoretical maximum. Consequently, we need to support burst lengths of more than 1, to deliver throughput close to the maximum possible throughput offered by HBM. To support burst operations, multi-flit networks are required. However, the reference NoCs used in our designs are single-flit networks, unsuitable for supporting burst operations. Therefore, we need to make modifications to the single-flit networks to support multi-flit operation.

In a multi-flit network, only the header flits carry the routing and memory information, as described in Section 3.1. Whereas the data flits that follow the header flits carry only the control bits along with the data. Therefore, the switches in our multi-flit network apply the routing algorithm to the header flit and route it to the desired output port. This routing decision must be applied to all the data flits that are part of the stream. To that end, when a routing decision is made for a header flit, a lock is placed on the requested output port. The switch, then uses this lock to route the following data flits to the output port request by the header flits. This lock remains in place until the data flit passes through the switch with the last bit set. To prevent interference from other streams, header packets at other ports are denied access to the locked output port when the lock is in place, and they are backpressured.

In Figure 3.2 we show a simplified architecture of the Pi switch used in BFT multi-flit NoC used in our designs. The critical difference between the Pi switch explained in Section 2.2.4 and the Pi switch used in our experiment, as shown in Figure 3.2 is the choice of the uphill port. For instance, packets that arrive at port L can take only U0 port to climb the network in the base design. Whereas, in our design, as explained in later sections, the choice of the uphill port depends on the memory address. This change requires a mux at the output ports U0 and U1 to choose between L and R. We use a similar architecture for single-flit Pi switches but without the lock system. Figure 3.3a shows the architecture of the single-flit Hoplite switch used in our design. From this figure, it can be observed that the S port of the switch is also shared with the PE output port. Hoplite NoC does not support multi-flit operation, as will be discussed in Section 3.5.3. Finally, In Figure 3.3b we show the architecture of the modified single-flit Hoplite switch that is used as the multi-flit ring switch in the Hybrid NoC. Observe that the multi-flit ring switch has two virtual buffers VC0 and VC1, which are required to ensure deadlock-free operation, as will be explained in Section 3.7.3.

## 3.3 SLR Crossing

Alveo U280 is comprised of three SLRs connected together by SLLs and Laguna registers. SLR crossing nets will incur significant time delay, if not between two Flip-Flops mapped to Laguna registers. This constraint is trivial to satisfy for a simple feed-forward data path. Where the SLR crossing net can be pipelined twice, and the pipeline registers can be

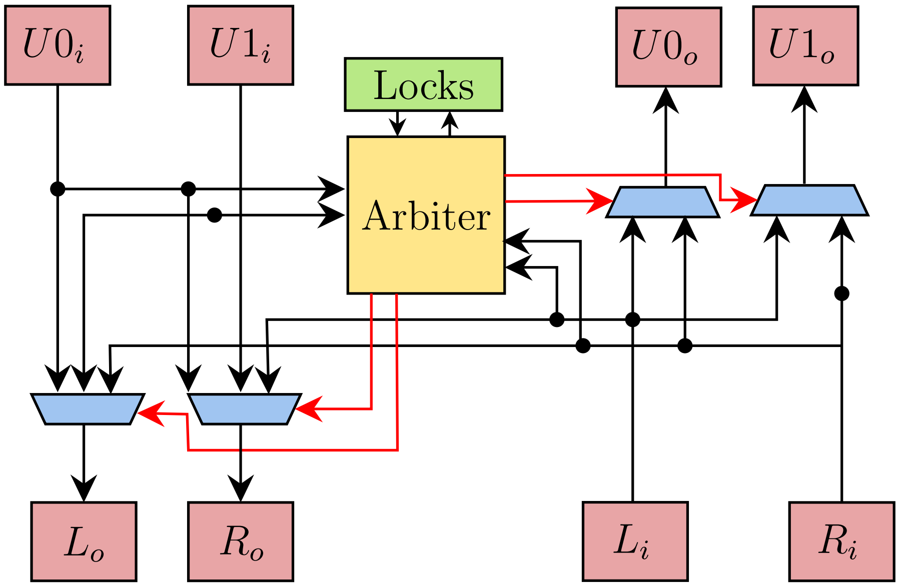
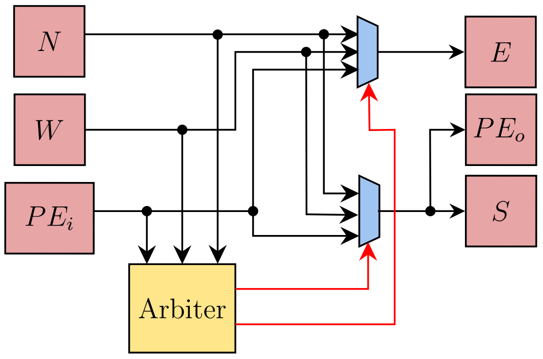
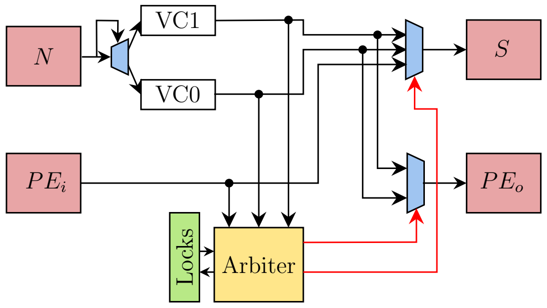


Figure 3.2: Architecture of the Pi switch used in Mulit-Flit BFT



(a) Architecture of the Single-Flit Hoplite switch



(b) Architecture of the Multi-Flit Ring switch using Hybrid NoC

Figure 3.3: Architectures of the Hoplite switch and the Ring switch used in Hybrid NoC

floorplanned to be placed on either side of the SLR boundary. Use of Vivado constraints such as *USER_SLR_ASSIGNMENT*, *USER_CROSSING_SLR*, and *USER_SLL_REG* also improve the reliability and speed of the placement and routing [52].

For instance, *USER_SLR_ASSIGNMENT* is used to assign leaf cells to different SLRs, whereas *USER_SLL_REG* is used to direct Vivado to map the specified register(s) to Laguna register(s). Similarly, *USER_CROSSING_SLR* constraint is used to direct on which net(s) the SLR crossings should happen.

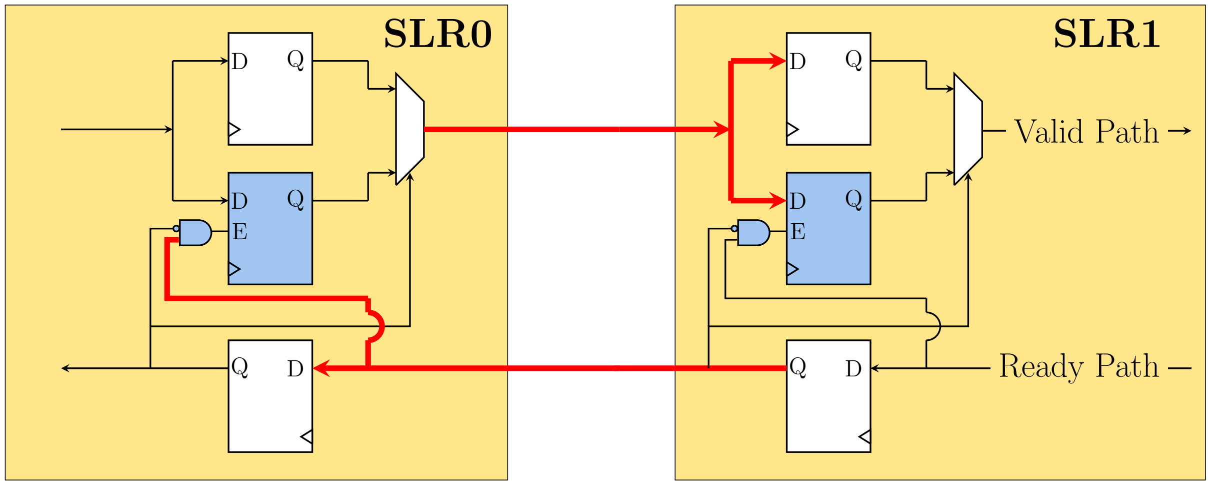


Figure 3.4: example of an SLR crossing for a Valid and Ready path. The need for a shadow register and a combinational circuit to maintain the functional correctness makes the SLR crossing to between a LUT and FF. For clarity, the data path is not shown, and it would look similar to the data path.

However, the SLR crossing gets complicated in the presence of a feedback path, as in the case of data paths using a Valid-Ready handshake. Simple pipeline stages will break the functional correctness of the data path. At the same time, a global stall would require the unregistered ready net to cross the SLR boundary, and the data path remains unregistered. We can retain functional correctness and introduce pipeline stages using shadow registers and a simple combinatorial circuit. In Figure 3.4 we depict an example of an SLR crossing on a Valid-Ready path from SLR0 to SLR1, using shadow registers. The shadow registers and the associated AND gates are highlighted in blue. However, with shadow registers, the SLR crossing path is always between a LUT (either a mux or AND gate) and a Flip-Flop in either path, as seen in the figure. Therefore, this SLR crossing using shadow registers will incur a significant time delay.

Instead, we use a First Word Fall Through (FWFT) Shift-Register-Lut (SRL) FIFOs with pipelined inputs and full registers to perform a FF to FF SLR crossing. With appropriate floorplan constraints, the data path and the ready path registers can be placed on either side of the SLR boundary. This ensures that the SLR crossing always happens between a FF to FF net, thereby utilizing the Laguna registers and decreasing the incurred path delay. In Figure 3.5, we depict this with an example of an SLR crossing using

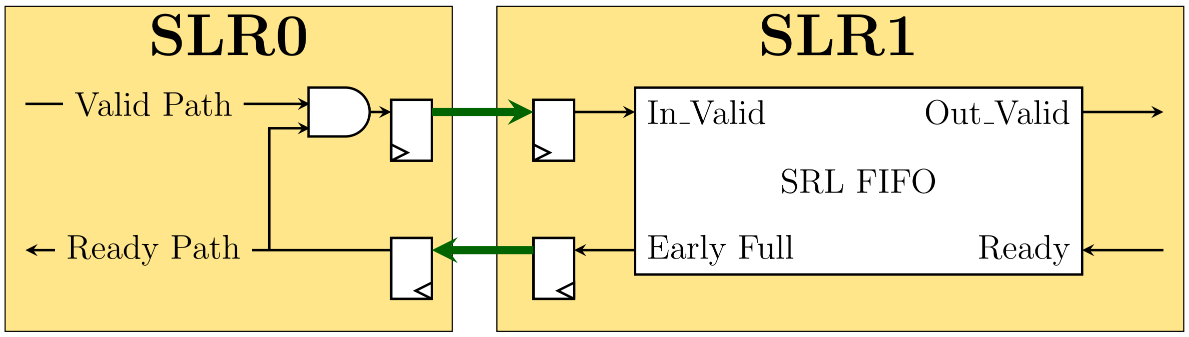


Figure 3.5: SRL Crossing of a Valid-Ready path using SRL FIFO and pipeline registers.

SRL FIFOs with two pipelined stages. The SLR crossing paths, shown in green, is always between two Flip-Flops and will not incur the enormous path delay penalty, unlike the path shown in figure 3.4. The functional correctness is maintained by accounting for the pipeline registers and issuing an early full based on the number of pipeline stages used. For instance, an SRL FIFO with a capacity of $C$ beats and $N$ pipeline stages (assuming $C > N$) must be configured to issue a full when $C - 2 * N$ beats are occupied.

## 3.4 HBM-NoC interface

The NoCs used in our designs and the HBM ports operate with different protocols. The NoCs either use a simple valid protocol or a valid-ready handshake protocol. Whereas the HBM ports, as discussed in Section 2.1, use the Full AXI3 protocol. Therefore, we need to convert the NoC protocol to AXI3 protocol in order to interface the NoC with the HBM ports. Furthermore, we also need to keep track of the PEs that have made the read requests, to route the read response from HBM back to the requesting PE. To meet these requirements, we use several FIFOs, as depicted in Figure 3.6, to interface our NoCs with the HBM ports.

The FIFOs play several key roles, and they are listed below.

1. Aid in converting the NoC level protocol to AXI3 protocol used by the HBM ports.
2. Allow Write address, Write Data, and Read Address AXI channels to progress in parallel and an unsynchronized fashion rather than in a lock-step fashion.
3. Aid in tracking PEs that have made read requests to route the read responses back to the requesting PEs.

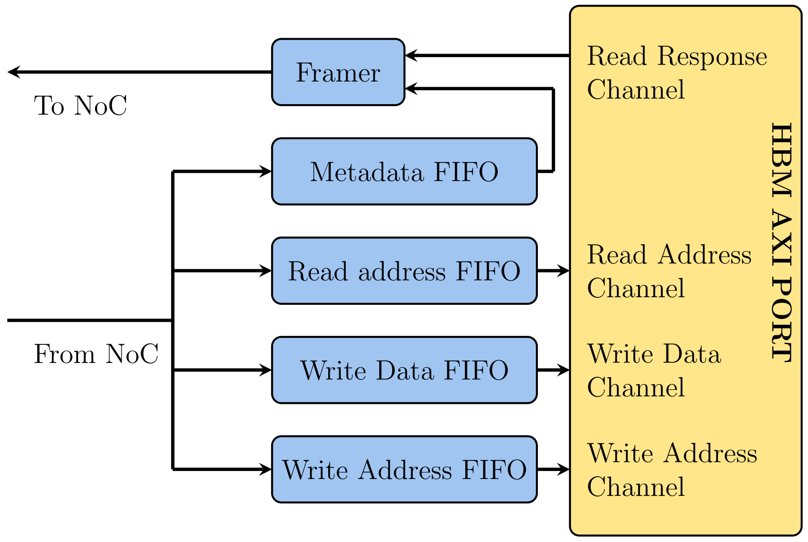


Figure 3.6: HBM-NoC interface module

4. Improve throughput by allowing more than one memory transaction to be queued up to be accepted by the HBM AXI ports.

The input from the NoC is parsed and based on the status of the Read-Write bit, the received address is directed to either the Write Address or the Read Address FIFO. If the Read-Write bit is set, the address is sent to the Write Address FIFO. If not set, it is sent to Read Address FIFO. In the case of writes, the data component from the flits is sent to the Write Data FIFO. For multi-flit networks, the data flits (non-header flits) do not bear any addressing or control information. Therefore, the HBM-NoC interface module retains the decision it made for the header flit, and it is applied to all subsequent flits until the entire stream, as indicated by the last bit, passes through. For read requests, only the header-flit is used to make the request. To prevent any loss of data from the NoC, the NoC is backpressured if any of the FIFOs in the HBM-NoC interface is full, and the backpressure is lifted only if an empty slot is available in all of the FIFOs.

The read response from the HBM must be routed back to the PE that requested it. To that end, the source address of the PEs making the read request is stored in the Metadata FIFO. When HBM responds to the read request, the read response and the head of the Metadata FIFO are sent to the Framer module. The Framer module then appropriately frames the return flits and injects the packet back into the network. For a single-flit network, the Framer concatenates the source address information from the Metadata FIFO with the read response from the HBM. For multi-flit packets, the header flit must bear the destination address, precede all the data flits, and be injected right when the first data flit is available. If the header packet is injected before the data flits are available, the path reserved by the header flit will be unutilized until the data flits are available, leading to poor link utilization. Therefore, the Frame module waits until the HBM responds with a valid data, and then it injects the constructed header packet into the network before framing and injecting the rest of the read responses from the HBM.

## 3.5 Adapting Hoplite for HBM

### 3.5.1 Choosing the dimension of the Hoplite NoC

Hoplite NoC has been used to interface PEs to HBM, as explained in Section 2.3.2. Therefore, as a starting point for our discussion, we use the Hoplite architecture used by Jan Gray in their design. The single-channel approach used by Gray is suitable for large PE counts, such as 225. However, for smaller PE counts, such as 32, the network is less flexible in terms of the usable dimensions. For instance, 32 PEs can be arranged in 1×32, 2×16, 4×8, 8×4, 16×2, and 32×1 dimensions. However, barring 1×32, with any other dimensions, all of the available 32 ports of the HBM cannot be used for interfacing. But with this dimension, all the traffic will travel along one ring, leading to higher congestion and poorer performance.

Instead, in our design, we use a multi-channel approach. Where several Hoplite networks operate in parallel, and the processing elements can ingress and egress packets through any channel via a fan-in and fan-out module, respectively. This multi-channel approach enables the use of all the available 32 HBM ports with a 4×8 dimension by employing four independent Hoplite channels. To account for the addition of channels, we extend the $R \times C$ dimension notation introduced in Section 2.2.5 to $R \times C \times Ch$, where $Ch$ is the number of channels, $R$ is the number of rows, and $C$ is the number of columns in each channel. Both 4×8×4 and 8×4×4 are viable dimensions. However, we choose 4×8×4 dimensions as it better suits the dimensions of the FPGA chip in Alveo U280.

As discussed in Section 3.1, the single-flit networks in our design use a flit of size 302 bits. Moreover, with our Hoplite network arranged in 4×8×4 dimensions, even if one row of our network is floor planned to a different SLR, there will be 64 SLR crossings. Therefore, in total, we need 19392 SLLs out of the available 23,040 SLLs, to meet the requirements for 64 SLR crossings. However, we use Vitis to implement and execute our design in Alveo U280. This prevents the use of 2880 SLLs, as explained in Section 2.4.1, and Vitis also consumes approximately 4000 SLLs, leaving only 16,160 for our network. Consequently, the SLL requirement for 32 vertical links (19,328) exceeds the available SLLs. To remedy this, we remove eight vertical links reducing the number of vertical links to 24, and we also reduce the number of PEs that we use to 24 as well. To implement this change, we remove two columns from our NoC and use only 24 HBM ports, making the dimension of the network 4×6×4. This change drops the SLL requirement to 14544 SLLs per SLR boundary, which is well within the available SLLs at each SLR boundary.

To interface an NoC with dimensions 4×6×4 and 24 PEs to 24 HBM ports, we use one additional row of Hoplite Switches, making the final dimension of our Hoplite NoC 5×6×4. In the first four rows of the network, only PEs are interfaced to the network, and each PE can ingress or egress packets to and from any of the 4 Hoplite channels. In the last row, the PE port of the Hoplite Switches is used to interface the network with an HBM port through the HBM-NoC interface, as shown in Figure 3.7. Since the memory packets get routed to their corresponding HBM port, no fan-in module is required in the last row. Likewise, the PEs can ingress from any channel. So, no fan-out module is required in the last row. To improve the readability of the figure, we have excluded the fan-out and fan-in modules and the loops around links.

### 3.5.2 Single-flit Operation

In the base design, PEs can inject their packets into any of the available channels. However, to route the memory packets to their corresponding home HBM port, the fan-out module chooses the channel based on the memory address. The memory packets are injected into the channel determined by the equation (**port_id** mod 4), where **port_id** is the most significant 5 bits of the memory address, and for a single flit network, it is encoded in bits 288 down to 284. Since the network is interfaced with HBM ports through switches only in the last row, the PEs set the destination X address, the upper 3 bits of the address, to point to the last row. Whereas the lower 3 bits, the destination Y address, depends on the **port_id** field of the memory address, and it is set as (**port_id**/4). Essentially, (**port_id** mod 4) calculates the channel to which the desired HBM port is connected, and (**port_id**/4) calculates the column within the channel.

Once injected into the correct channel by the fan-out module with the proper X and Y destination addresses, the network routes the memory packet like any other packet. Due to the network architecture and routing algorithm, the injected memory packet is routed to the AXI port corresponding to the specified memory address. Consequently, eliminating the traffic in the lateral links and circumventing the throughput bottleneck introduced by it. In Figure 3.7, we highlight the path taken by a memory packet injected into the network by PE10 bearing the memory address 170000000$h$. The **port_id** field of this memory address is 23. Accordingly, the network must route this memory packet to HBM AXI port 23. To that end, the PE needs to set the destination Y address to 5 and inject the packet in the 4th channel. Then, the memory packets get routed like any other packet, and due to the architecture of the network, it is routed to HBM AXI port 23.

Apart from the homing of memory packets, this network retains the PE to PE communication feature of the Hoplite NoC. The packets intended for other PEs can be easily distinguished from the memory packets by their destination X address. Since all HBM-NoC modules in the network are interfaced in the last row, any packet with a destination address other than that last row is deemed to be a PE to PE packet. The PEs can use the network either as a single channel Hoplite by setting the **port_id** field of the memory address to 0. Alternatively, they can choose the channel by setting the **port_id** field to be 1, 2, or 3. While this setup works without any livelock or deadlock for writes, it gets livelocked during reads.

### Livelock

In our Hoplite NoC, the request packets and the read response packets share the same physical link. Similarly, the South port of the Hoplite switch is shared with the input port of the PE. These two architectural quirks of the Hoplite NoC create a possibility of livelock.

To illustrate this, consider an example depicted in Figure 3.8 where we have shown a vertical link of a toy Hoplite network with two rows and one column. The horizontal links do not play any part in this livelock example, and therefore, they are omitted from our figure. Also, assume that the Metadata FIFO in the HBM-NoC interface module is full. Due to the homing feature and DOR used in the Hoplite network, memory packets (PE $\rightarrow$ HBM) that arrive at the north port of Switch 1 will always request to route to the interface module. So, packet B desires to be routed to the interface module (path ①). However, the Metadata FIFO is full, and the interface module indicates to the switch that it is not ready to accept any new packets. Therefore, packet B is deflected to the south port instead (path ②). Meanwhile, packet C from the Metadata FIFO in the HBM-NoC interface module, present at the PE port of Switch 1, wants to route to the South port (path ③). But, packet B at the north port has higher priority than packet C, and packet B wins the contention. As a result, packet C is rejected, and the PE port is backpressured. In the next cycle, packet B is replaced by packet A from Switch 0 (path ④), and the whole process repeats again.

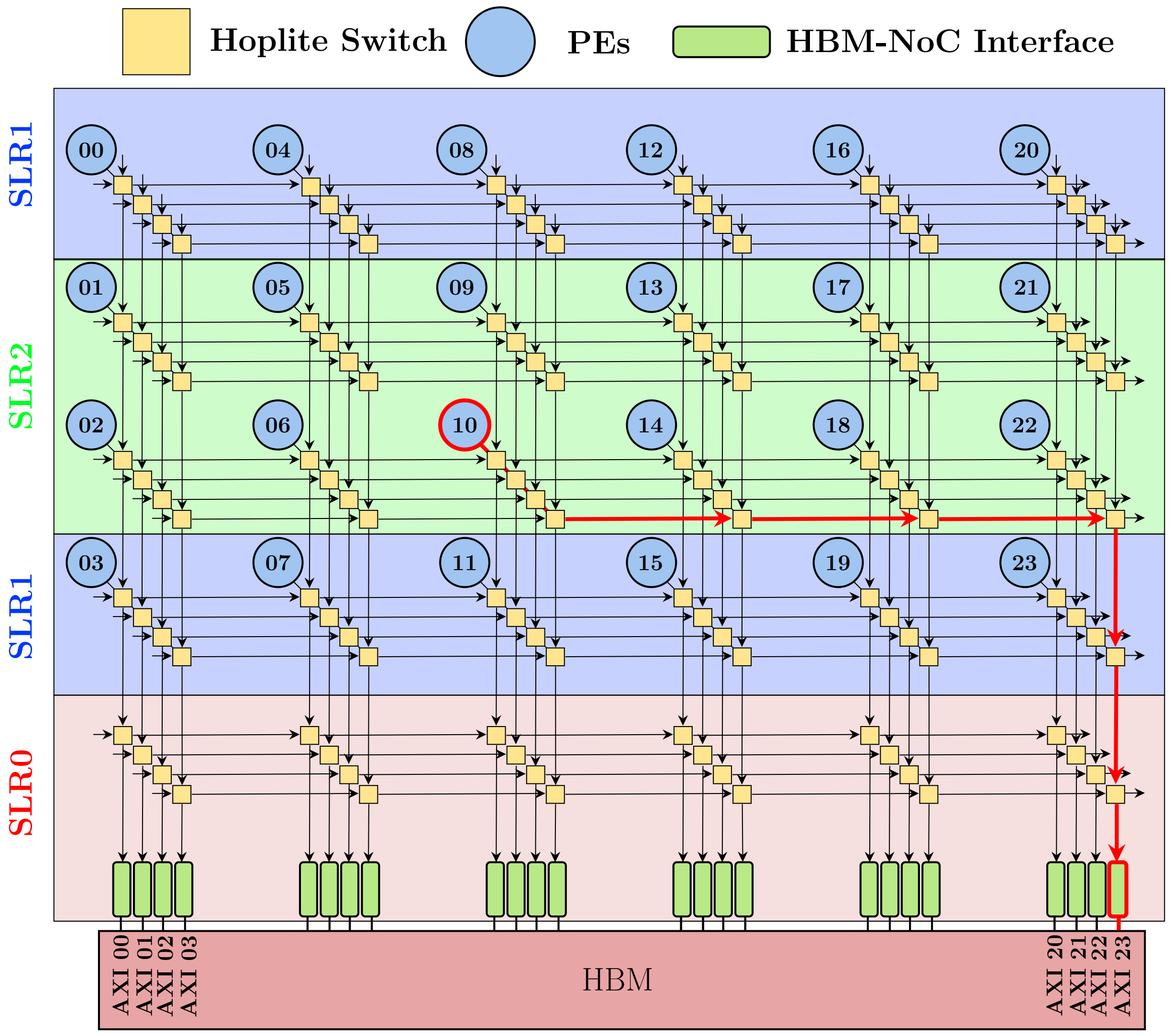


Figure 3.7: PEs interfaced to HBM using Hoplite NoC. Modules in the RED region are assigned to SLR0, and it includes HBM-NoC interface modules and the HBM ports along with the highlighted switches. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The fan-in and fan-out modules through which the PEs ingress and egress packets are not shown. Similarly, the vertical and horizontal loops around links are also not shown here for readability. The path highlighted in red is taken by memory packets injected by PE 10 with memory address 170000000$h$

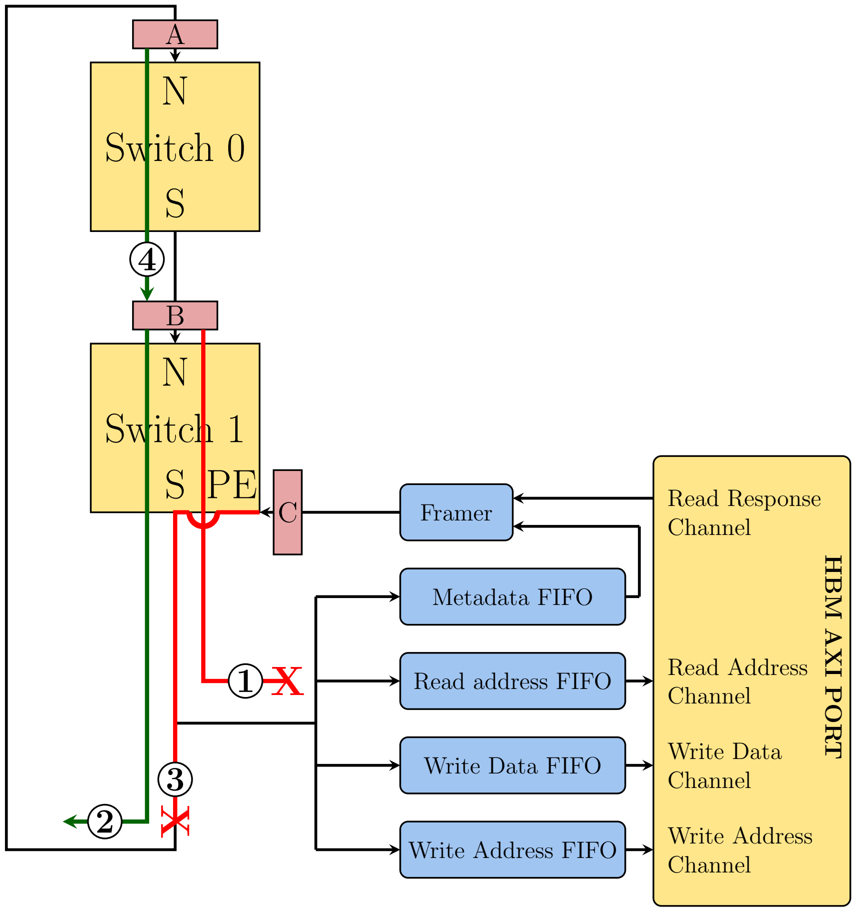


Figure 3.8: Example of a livelock. ① Packet B wants to route to the HBM-NoC interface module, but it is denied because Metadata FIFO is full. ② Instead, it is deflected to the south port. At the same time, ③ packet C from the HBM-NoC module wants to leave the Metadata FIFO by exiting the through south port. However, ② has higher priority than ③, therefore, packet C is rejected, and the PE port is backpressured. At the same time, ④ packet A from switch 0 arrives at the north of switch 1, and the whole process repeats again.

This situation leads to a livelock between packets at the north port and packets at the PE input port of switches in the last row. The north packets cannot enter the HBM-NoC interface module since the Metadata FIFO is full, and are routed to the south port instead. But, the only way to clear the Metadata FIFO is by routing packets at the PE input port through the south port, yet they are denied the south port because the north packets, rejected by a full Metadata FIFO, are deflected to the south port. This process will lead to livelock if the number of request packets travelling in the vertical link that wants to sink into the interface module equals the number of rows in the network. Because then under no circumstances can the Metadata FIFO clear out through the south port because the deflected north packets constantly occupy the south port, leading to livelock. This scenario occurs only when the HBM-NoC interface modules want to inject packets into the network, as in the case of read responses. Write responses are ignored in our design.

We resolve this livelock issue by limiting the number of pending read requests that a PE can have and accordingly sizing the FIFOs in the HBM-NoC interface module to handle the worst case. This ensures that the number of read request packets in the column is always less than the number of rows. In our design, we have limited the number of pending read requests per PE to 10. Moreover, in the worst-case scenario, where all the 24 PEs access only one port, the FIFOs must hold 240 flits. Therefore, we size our FIFOs to hold 256 flits, the nearest power of two larger than 240. With this setup, the FIFOs can absorb all the read requests, breaking the circular dependency and preventing livelock.

### 3.5.3 Multi-flit

As described in the Section 3.2, we have to incorporate a lock system in our switches to support multi-flit routing. Moreover, a simple deflection-based routing is not ideal for a multi-flit network. With deflection-based routing, the flits of the same stream can get dispersed, leading to interference from other streams and out-of-order arrival at the destination. Therefore, we need to replace the deflection-based routing with lightweight flow control with backpressure. However, this can lead to deadlock in the horizontal links.

Consider the example shown in the Figure 3.9. Here we have three columns of Hoplite switch, with PEs A, B, and C. Assume that PE A is injecting a stream of flits (stream A) that want to turn south at Switch 2. Similarly, PE B injects a stream of flits (stream B) that want to turn south at Switch 0, and PE C flits (stream C) want to turn south at Switch 1. In this scenario, all the flits want to travel east and immediately block each other because of the lock system. In a multi-flit network, a stream is given the lock to the desired port at any switch until the last flit passes through the switch. In our scenario, stream A receives the lock of the east port of Switch 0. Likewise, stream B and stream C receives the lock of the east port of switch 1 and 2, respectively. This leads to circular dependency along the horizontal link, and the streams cannot progress due to the lock in place. The only way to remove the lock is if the entire stream passes through the switch, leading to deadlock.

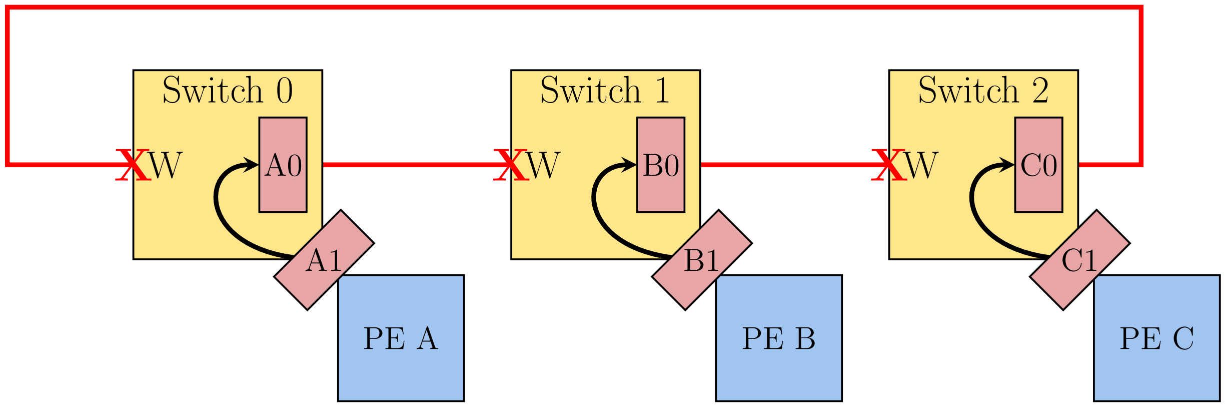
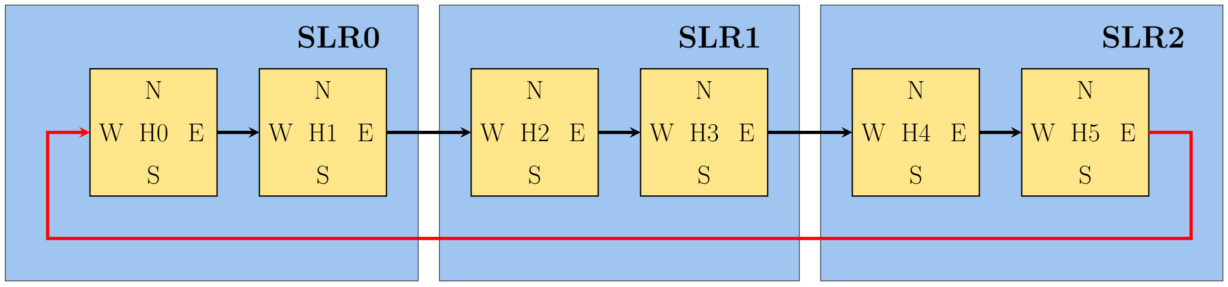


Figure 3.9: Example of a deadlock in the Horizontal Link. In a multi-flit Hoplite, when all flits in the horizontal link want to travel east, it leads to circular dependency due to the lock system.

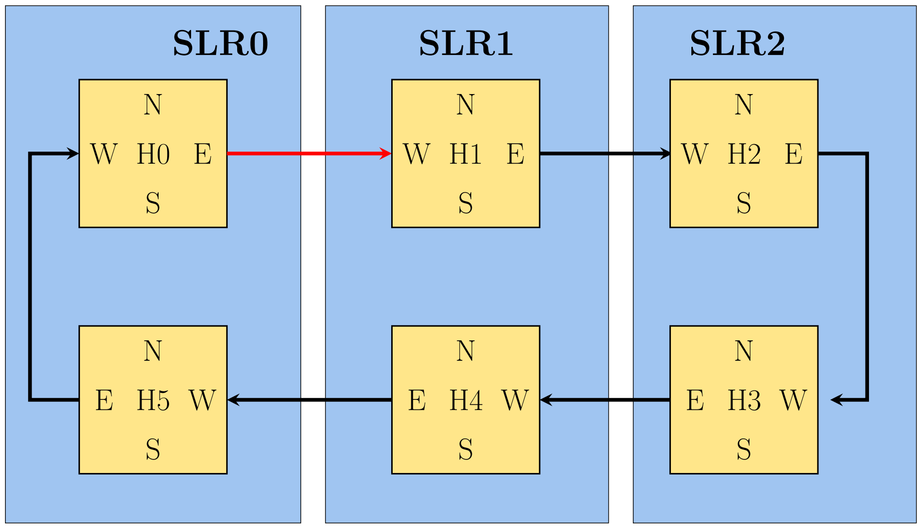
One possible way to fix this would be to introduce Virtual channels along the horizontal channels. But, this scenario can also rise along the vertical channels, requiring the use of Virtual channels at every switch, making the NoC expensive in terms of resource utilization. Therefore, we do not implement the multi-flit feature in Hoplite networks. In our experiments, the Hoplite network is limited to single-flit experiments.

### 3.5.4 Floorplan

One of the objectives of this thesis is to support chip-spanning design, and to that end, we floorplan our Hoplite NoC to span the chip. The 2D torus topology of Hoplite makes it simple to map it to a rectangular chip. However, the vertical and horizontal loop around the link that completes the torus spans the entire length and breadth of the FPGA chip, as shown in Figure 3.10a. The longest path (highlighted in red) lies between H5 and H0, and it spans the entire chip, crossing all three SLR boundaries. Without pipelining, these links become the critical path, and due to the large path length, they incur enormous path delay, increasing the achievable clock period. One solution to this problem is to pipeline these links. However, pipelining is not the most resource optimal solution for wide link bitwidth. With careful floorplanning, the worst-case wire length can be significantly lowered by folding the torus, as shown in Figure 3.10b. The longest path is now between any two consecutive switches, and in the figure, we highlight one possible critical path. This folding of the torus reduces the path delay considerably, without any additional resources.



(a) Unfolded torus floorplan. The critical path from H5 to H0, highlighted in red, spans the entire chip crossing all the SLR boundaries



(b) Folded torus floorplan. The critical path, highlighted in red, is considerably smaller than the unfolded torus floorplan, and this achieved without adding any pipeline registers.

Figure 3.10: Unfolded vs Folded torus floorplanning.

In the figure, 3.7 we show a top-level floorplan used for the Hoplite NoC. The folding of the vertical links can be observed from this figure. However, we do not show the folding in the horizontal for improving the readability. Switches from the last row and the HBM-NoC interface module are assigned to SLR0, and they are left unfolded. Whereas SLR1 and SLR2 get two folded rows with 12 PEs each. As mentioned earlier, this floorplanning results in 14544 SLLs at both the SLR boundaries. In Figure 3.11 we show the post Place and Route device view of the Hoplite NoC implementation (without PEs) in Alveo U280.

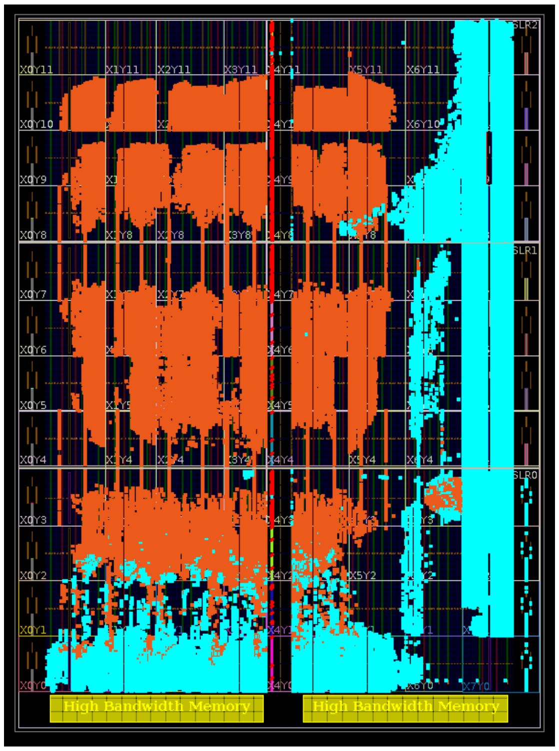
In the left Figure 3.11a we show the leaf cells of the Hoplite NoC and HBM-NoC interface in orange, and the leaf cells of other supporting modules instantiated by Vitis are shown in blue. In the right Figure 3.11b we show the congestion heat map of the same. Barring some high congestion in the static region used by Vitis and SLR0, our design suffers little congestion. The congestion SLR0 can be attributed to the unfolded last row of the Hoplite NoC. Without any constraints, the Vivado places and routes the leaf cells closer together, leading to higher congestion. Nevertheless, since the last row connects to the fixed location of HBM ports, it cannot be folded. We tabulate the resource utilization and the maximum frequency with and without different PEs in the result section.

## 3.6 Adapting Butterfly Fat Tree for HBM

### 3.6.1 Choosing the configuration

As explained in Section 2.2.4, BFT with lightweight flow control [32] is the starting point of our design. The BFT is built using Pi and T switches, and the bisection bandwidth of the network can be configured by varying the type of switch used at each level of the network. There are four possible configurations - XBAR, MESH0, MESH1, and TREE. In the XBAR configuration, all the levels of the tree are built using Pi switches. This configuration offers the highest bi-section bandwidth of $O(N)$ and, on the other end of the spectrum, is the TREE configuration. In the TREE configuration, all levels of the network are built using T switches, and it offers the lowest bi-section bandwidth of $O(1)$. MESH0 and MESH1 are built using varying Pi and T switches at each level, and they offer bisection bandwidth of $O(\sqrt{N})$.

One consequence of using T switches is the reduction in the number of uphill ports. For instance, only T switches are used in the TREE configuration, and at each level, the number of the uphill port is halved. As a consequence, we would have only one uphill port at the topmost level of the network. On the other hand, the XBAR configuration preserves the number of the uphill port at every level, which makes it a candidate for interfacing 32 PEs with 32 HBM ports. For MESH configurations (MESH0 and MESH1), the number of uphill ports at the topmost level will not match the number of PEs interfaced with it. For example, in a MESH1 BFT with 32 PEs, only 8 uphill ports would be available (similar to Figure 2.9b). Therefore, we need 4 channels of the BFT in MESH configurations to interface with 32 HBM ports to meet our objective. In Table 3.1 we tabulate the number of Pi and T switches required to meet our objective of interfacing 32 PEs to 32 HBM ports. Clearly, MESH0 and MESH1 consume more switches than XBAR without offering any throughput


(a) The leaf cells highlighted in orange are instantiated by our design. Whereas, the-leaf cells highlighted in blue are from modules instantiated by Vitis.

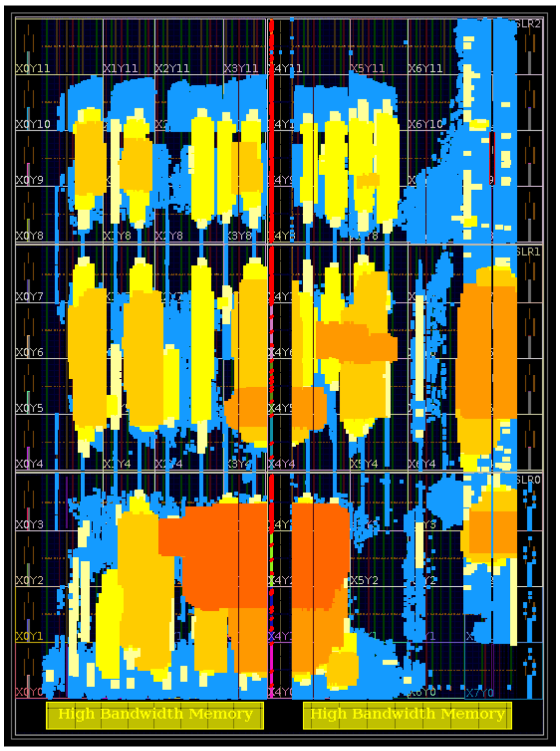

(b) Congestion Heat Map. The regions highlighted in blue suffer no congestion, and the regions with congestion level 1 are shown in yellow. As the congestion level increases, the colour becomes redder. The highest congestion in this network is level 5 highlighted in orange.

Figure 3.11: Post Placement and Route Device View for Hoplite NoC without PEs.

advantages. Therefore, in this thesis, we use BFT in the XBAR configuration in all of our designs.

To interface the BFT NoC with HBM AXI ports, we use the uphill ports at the topmost level of the network. They are connected to the HBM ports via the HBM-NoC interface module, whereas the PEs are connected to the leaf-level ports, as shown in Figure 3.12. Thus, the memory requests always ascend the network, and the responses always descend the network. While this BFT network can be used to interface with 32 HBM ports without exceeding the available SLL resources, to make valid and fair comparisons between Hoplite

Table 3.1: Number of Switches required to interface with 32 HBM port using BFT in various configurations

| **NoC Topology** | **Switch Config** | **Switches per Channel** | | **Total Switches** | |
|---|---|---|---|---|---|
| | | **Pi** | **T** | **Pi** | **T** |
| XBAR | Pi-Pi-Pi-Pi-Pi | 80 | 0 | 80 | 0 |
| MESH1 | Pi-Pi-T-T-Pi | 36 | 24 | 144 | 96 |
| MESH0 | Pi-T-Pi-T-Pi | 28 | 24 | 112 | 96 |
| TREE | T-T-T-T-T | 0 | 31 | 0 | 961 |

and BFT, we drop eight PEs and eight HBM ports. Therefore, we only use 24 PEs interfaced with 24 HBM ports in our experiments unless stated otherwise.

### 3.6.2 Single-flit

In the base BFT, the upward routing freedom is limited. The choice of the uphill port is strictly based on the downhill port at which the packets arrive, i.e., packets wanting to ascend the network that arrives at the downhill L port can only route to the uphill U0 port ( L $\rightarrow$ U0). Similarly, ascending packets that arrive at R port only route to the U1 port ( R $\rightarrow$ U1). This restriction simplifies the routing algorithm, reduces the resources count, and ensures deterministic routing. However, it also prevents the NoC from routing the memory packets to their corresponding home AXI ports. So, to provide homing, the ascending memory packets can take either U0 or U1 based on the HBM memory address carried by the packet. For instance, at level $l$ of the network, the uphill output port is chosen based on the bit $l + 1$ of the **port_id**. Unlike other levels, at the topmost level, bit 0 determines the uphill port. If the bit is set, port U1 is chosen. Otherwise, Port U0 is chosen. For responses descending the network, no change in the routing algorithm is required. The choice of downhill port is also limited, and it depends on the packet's destination address supplied by the Metadata FIFO. At level l, the bit l of the destination address determines the downhill port. If it is set, port L is chosen. Otherwise, R is chosen. In Figure 3.12, we show an example of this routing. PE 23 injects a memory packet with the HBM memory address set to 70000000$h$. Since the **port_id** is set 00111$b$, at level 0 and level 1 of the network, the packet ascends through port U1. At level 2 and level 3, the packet uses the uphill port U0. Finally, at the topmost level, the packet uses port U1, sinking into HBM AXI port 7 through the interface module. For responses, the packets take the same route, but it descends through the network instead.

BFT also retains the PE to PE communication capabilities. Similar to the Hoplite NoC, the destination address distinguishes memory packets from other packets. In BFT, the HBM ports are interfaced at the top. Accordingly, the destination address of the memory packets is chosen so that the network always routes it uphill towards the HBM ports. It can be done by setting the destination address as $N$ for a network with $N$ PEs. In our case, we set the destination address to 32 for memory packets. PEs can communicate with other PEs by setting the packet's destination address to any value up to 31.

In BFT, unlike the Hoplite network, the request and responses travel in the different physical channels, and the switches use a lightweight flow control with backpressure instead of deflection-based DOR. So, with BFT, there is no possibility of circular dependencies leading to livelock during reads. Hence, we do not have to limit the number of pending read requests in the network or increase the size of the FIFOs in the interface module, as we did with Hoplite NoC.

### 3.6.3 Multi-flit

The original BFT network only supports single-flit routing. To support multi-flit routing for burst lengths longer than one, we incorporate the locking system explained in Section 3.2 in our BFT switches. The resulting Pi switch is shown in Figure 3.2. Unlike the Hoplite network, the BFT topology has no loop around links, and the read responses travel in a separate physical channel. These characteristics of the BFT, and the underlying routing algorithm used by it, ensure deadlock-free operation for multi-flit packets. Therefore, BFT is an ideal candidate for interfacing PEs that require burst operations with HBM. To that end, we conduct our multi-flit synthetic and application benchmarking experiments with BFT in XBAR configuration.

### 3.6.4 Floorplan

As per objective, we floorplan the BFT NoC with 24 PEs to span the chip across all three SLRs in the Xilinx Alveo U280 as shown in 3.12. For the sake of readability, only the SLR placement is shown, and the placements within SLR are not shown. Since most of the HBM subsystem modules and HBM-NoC interface modules need to be assigned to SLR0, we wanted to minimize the number of NoC modules assigned to SLR0. However, this proved to be a challenging task due to the hierarchical nature of the BFT.

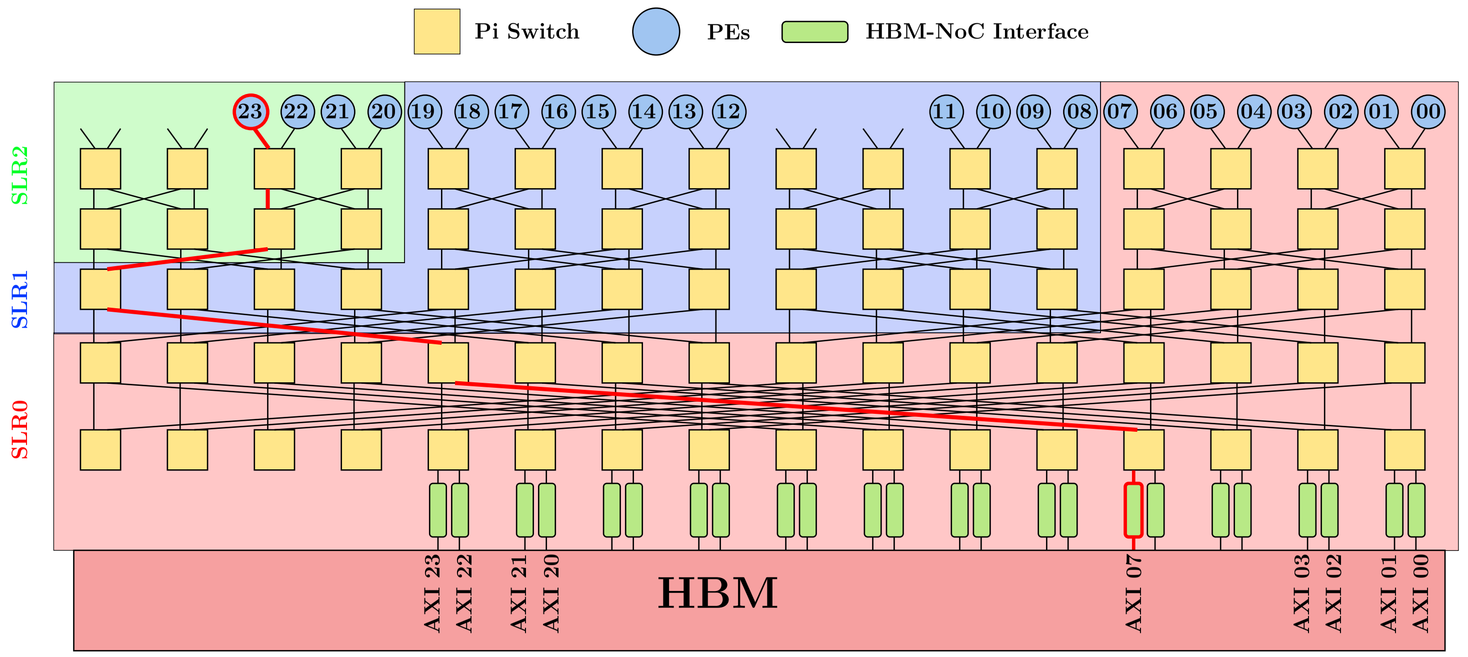
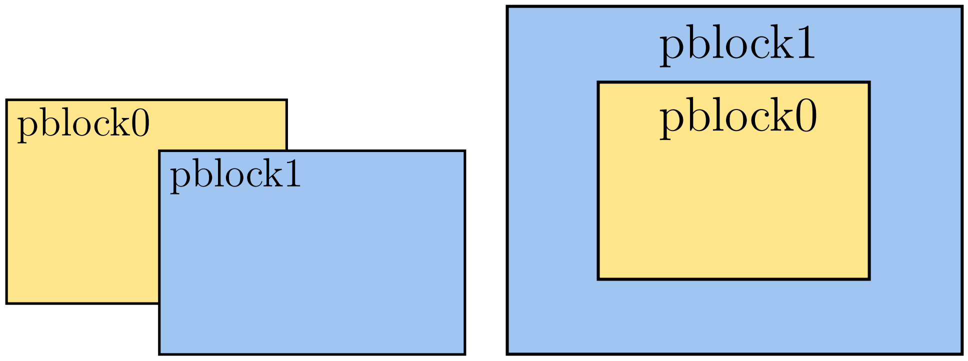


Figure 3.12: PEs interfaced to HBM using BFT. Modules in the RED region are assigned to SLR0, including HBM ports. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The path highlighted in red is taken by memory packets injected by PE23 with the memory address set to 70000000$h$. Note that the image is rotated for readability. The PEs are connected to the leaf nodes at the bottom of tree, and the HBM ports are connected to the topmost uphill ports of the tree.



(a) Overlapping Pblocks (b) Nested Pblocks.

Figure 3.13: Difference between a overlapping pblocks and nested pblocks.

In Vivado, the users can direct the placement of their design through placement blocks (pblock). First, the users assign FPGA resources from a rectangular area to a pblock. Then, the tool can be forced to place selected modules and cells from the design (dictated by the user) within the pblock. However, there are certain restrictions on pblocks when it comes to overlapping multiple pblocks. Vivado does not accept partially overlapping pblocks. Instead, it forces a parent-child relationship between them, where one of the overlapping pblock called the parent must completely envelop the other pblock, called the child pblock, as shown in Figure 3.13. For BFT, this results in a highly nested pblock structure. In Figure 3.14, we show an example floorplan for a BFT with eight PEs. It can be seen that, for a valid pblock assignment, pblocks for level 2 switches must encompass pblocks assigned for level 1 switches, and pblocks for level 1 switches must encompass pblocks assigned for level 0 switches and PEs. These nested pblock assignments result in a placement where most of the leaf cells assigned to the pblock end up getting placed near the corner of the rectangle, closest to the center of the chip—resulting in higher utilization and higher congestion along the vertical axis of the chip. Considering the floorplanning restriction of Vivado and the SLL availability, 8 PEs were assigned to SLR0, 12 PEs were assigned to SLR1 and, and 4 PEs to SLR2, as shown in 3.14. For the single-flit network, this requires 14544 SLLs at the SLR0-SLR1 boundary, and 4848 SLLs at the SLR1-SLR2 boundary, well within the available SLLs. The SLL requirements reduce further for multi-flit network due to the smaller flit size. As a result, only 12528 SLLs are required at the SLR0-SLR1 boundary, and 4176 SLLs are required at the SLR1-SLR2 boundary.

In Figure 3.15 we show the post Place and Route device view of the BFT single-flit NoC implementation (without PEs) in Alveo U280. In the left 3.15a we show the leaf cells of NoC and FIFOs in orange, and the leaf cells of other supporting modules instantiated

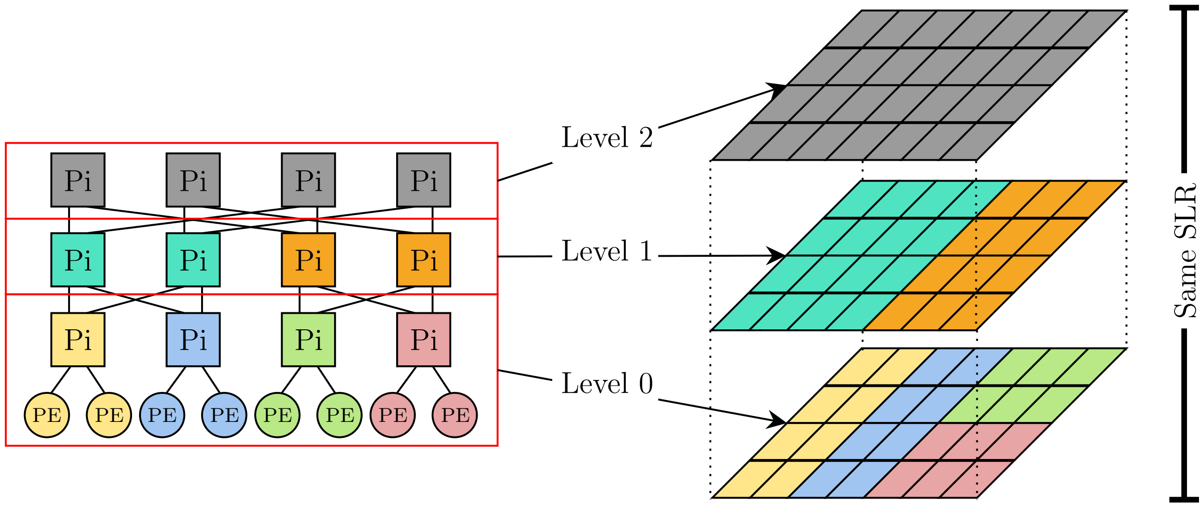


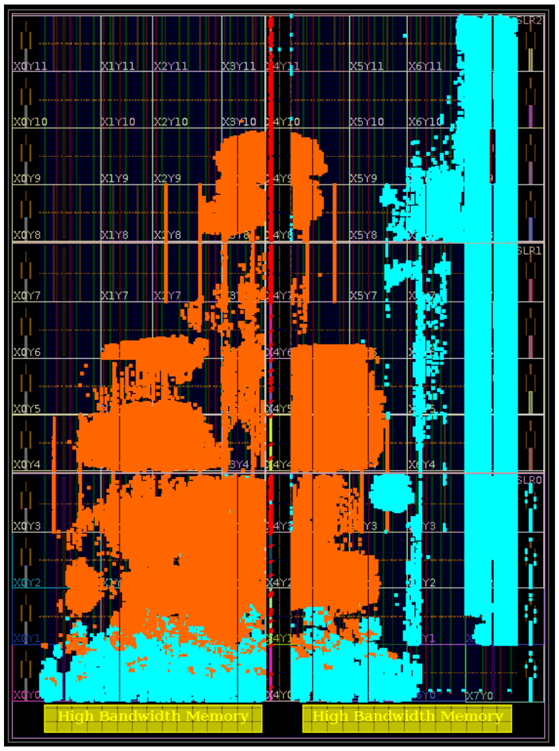
Figure 3.14: Example floorplan for a BFT with 8 PEs. On the right, we have a coloured BFT with 8 PEs. On the left, we show the pblock assignments on one SLR with 4×7 Clock Regions, where the colour of the pblocks matches the Pi and PE modules in the BFT NoC shown on the left.

by Vitis are shown in blue, and in Figure 3.15b we show the congestion heat map of the same. For this figure, it can be observed that due to nested pblocks and the placement behaviour of Vivado, most of the leaf cells are placed towards the center of the spine of the device, leading o high congestion on the routing resources, even without any PEs. With PEs, this congestion problem is further worsened, and the operating frequency drops, as shown in the results section.

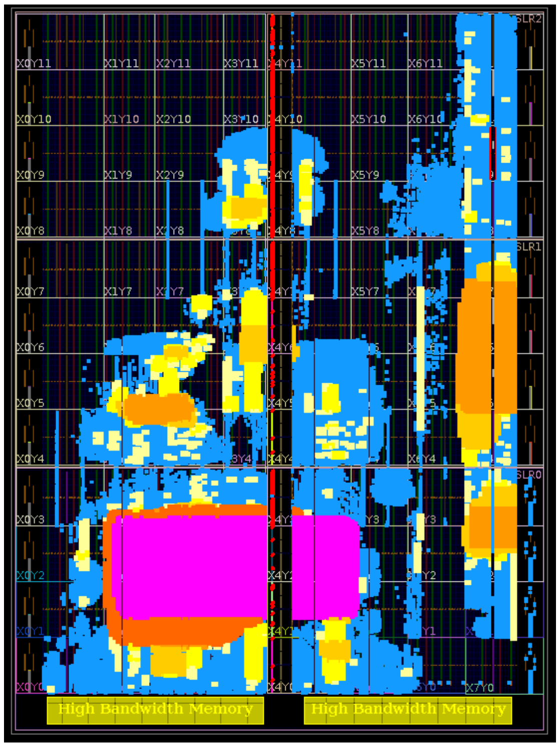
## 3.7 Hybrid

### 3.7.1 Need for a Hybrid NoC

Both the Hoplite and BFT NoC have shortcomings that need to be addressed. For example, Hoplite NoC suffers from livelock and deadlock. The livelock problem is circumvented by limiting the number of pending read requests in the network. However, it degrades the throughput performance. Similarly, the Hoplite NoC is non-ideal for multi-flit operations. The inclusion of backpressure and a lock system, required for ensuring interference-free multi-flit routing, leads to circular dependencies along both the horizontal and vertical



(a) The leaf cells highlighted in orange are the NoC and FIFOs cells. The leaf cells highlighted in yellow are from modules instantiated by Vitis.



(b) Congestion Heat Map. The regions highlighted in blue suffer no congestion, and the regions with congestion level 1 are shown in yellow. As the congestion level increases, the colour becomes redder. The maximum congestion level observed with the single-flit BFT network is 6, shown in pink.

Figure 3.15: Post Placement and Route Device View for BFT single-flit NoC without PEs.

links. Hence, requiring expensive virtual channel buffers and complicated flow control at every switch to ensure deadlock-free operation. On the other hand, BFT NoC does not suffer from livelock or deadlock.

However, BFT is inflexible in terms of floorplan due to the network's topology. With Hoplite, the topology of the network allows for PEs to be arranged in different dimensions and moved across SLR without changing the SLL requirement. This flexibility simplifies floorplanning and prototyping with Hoplite network, where the number of PEs in the network and the number of PEs assigned per SLR can be changed without significant changes in the RTL and without incurring time cost. The hierarchical topology of BFT makes it rigid when it comes to the arrangement of PEs across the chip. Moving PEs from one SLR to another entails several RTL and constraint changes. And, it also changes the number of SLLs required, making it difficult to explore different floorplan possibilities with BFT. Furthermore, spreading the leaf cells of the BFT along the breadth of the chip to combat the congestion, as described in Section 3.6.4, requires more pipelining stages, and in some cases, it worsens the congestion problem. So, both BFT and Hoplite offer some unique advantages, but they also have several shortcomings.

### 3.7.2 Architecture

This section proposes a Hybrid NoC architecture that combines BFT and Hoplite by borrowing features from both networks to address their respective shortcomings. In Hybrid NoC, we borrow the vertical links from the Hoplite NoC to provide inter-SLR communication, and we use BFTs to provide intra-SLR communication. There are two perspectives to view the architecture of the Hybrid NoC. In the first view, the Hybrid NoC can be seen as a Hoplite NoC, with the horizontal link substituted with rows of BFTs. In the other view, the Hybrid NoC can also be seen as a BFT NoC, with the top two levels of Pi switches replaced by Ring NoCs connecting the bottom three levels of the BFT. Instead of having one extensive network, in the Hybrid NoC, there are several rows of BFT NoCs connected by Ring NoCs.

Our objective is to interface 32 PEs with the 32 HBM ports. To that end, in our Hybrid NoC, we use 32 vertical ring NoC that span the entire chip. The ring NoCs use a switch similar to the Hoplite switch, barring one key difference. In the Hoplite switch, the south port was also used as the input to the PE. However, this complicates the routing algorithm for multi-flit packets. Therefore, switches in the Ring NoC use a separate PE input port to sink inputs into PEs. Furthermore, the horizontal links are replaced by the BFT. Therefore, the East and West port are also removed.

The 32 PEs are grouped into four rows of BFTs, each with eight PEs. A row of BFT will have eight ports available at its topmost level for interfacing with the 32 rings. So, we use a fan-out and fan-in module, as employed in the Hoplite NoC, to connect one top-level port to four rings. A switch at the topmost level of the BFT can ingress and egress packets into any of the connected four rings. Similar to Hoplite, the choice of the ring is dictated by **port_id** field of the header flit. For memory packets, the fan-out module chooses the channel determined by the equation (**port_id** $\bmod 4$), whereas for other types of packets, the PEs can choose the ring by varying the **port_id** field of the header flit.

Each vertical ring has four endpoints for the four rows of BFT, like the Hoplite NoC, one more endpoint is added for interfacing with HBM ports through the HBM-NoC interface module. So, the first four endpoints connect to the rows of BFT, and the fifth endpoint connects to the HBM-NoC interface module. However, using all 32 rings will require 16704 SLLs, which is more than available after accounting for Vitis. Therefore, we need to drop a few vertical links to fit the Hybrid NoC in the chip. To remain consistent with the BFT and Hoplite NoC, we drop eight vertical rings and use only 24 vertical links to interface 24 PEs with 24 HBM ports.

### 3.7.3 Multi-flit Operation

The Hybrid NoC does not support single-flit operations, and it is left as future work. All the switches, fan-in, and fan-out modules in the NoC use the lock system described in Section 3.2, to support multi-flit routing. The ring switch used in the Hybrid NoC is shown in Figure 3.3b, and the Pi switch used is the same as the Pi switch in BFT. Similar to Hoplite, the address field is split into components - X address and Y address. The upper three bits, assigned to the X address, indicate the destination row BFT. Whereas the lower three bits, assigned to the Y address, indicate the position of the PE within that row of BFT. For memory operations, the PE must set the X address to point to the last row, whereas the uphill through the BFT is purely dependent on the memory address. Therefore the Y address is irrelevant and can be left as zero. Once injected into the network, the memory packet climbs the BFT, as described in Section 3.6, routed towards the ring that connects to the desired HBM port. At the topmost level, the fan-out module injects the packets into the ring determined by the equation (**port_id** $\bmod 4$). Then, the memory packet travels down the ring and reaches the desired HBM AXI port as defined by the memory address. In Figure 3.17, we highlight the path taken by a packet injected into the network by PE5 bearing the memory address $60000000h$ in red. Since the **port_id** is set as $00110b$, at level 0 and level 1 of the network, the packet ascends through port U1. At level 2, the packet uses the uphill port U0 to sink into the 2nd ring ($6 \bmod 4$) via the fan-in module. Then, the ring carries the packet to the last row and the packet sink into the 6th HBM port. For responses, the packets loop around the same ring and sink into the first BFT row, which carries the packet to its destination. The PEs can use the network for communication with other PEs by varying the X and Y addresses. Similarly, for PE to PE packets, the channel can be chosen by varying the **port_id** bits.

There are a few advantages to this topology. In Section 3.5.3 we demonstrated that there is a possibility of deadlock forming in the horizontal links of the Hoplite NoC. By substituting the horizontal links with mini-BFTs, we ensure deadlock-free operation in the horizontal axis for multi-flit packets without the use of virtual channels. Furthermore, this topology retains the floorplan flexibility of the Hoplite NoC that was absent in BFTs. Adding and removing PEs in the Hybrid NoC is similar to Hoplite. It can be done by adding or removing rows from the network. Furthermore, moving PEs from one SLR to the other is also similar to the Hoplite NoC. The number of SLL requires does not change when a row of PEs is moved from one SLR to another, and it requires straightforward floorplanning with pblocks, with little to no change in RTL design.

#### Deadlock Issue

With multi-flit routing, there is a possibility of circular dependencies forming along the rings similar to the livelock issue of Hoplite explained in Section 3.5.2. Consider the example shown in Figure 3.16. Assume that the Metadata FIFO is full, and packets B, A want to sink into the HBM-NoC interface module. Since the Metadata FIFO is full, packet B is blocked (path ①), and the north port of Switch 1 (connected to the south port of Switch 0) is backpressured (path ②). This results in packet A getting blocked (path ③), and the north port of Switch 0 is backpressured (path ④). Now, HBM responds, and the Framer wants to inject packet C into the network, which will free up a slot in the Metadata FIFO and allow packet B to progress. The only way for packet C to exit Switch 1 is through the south port. However, the south port is backpressured, and packet C is blocked (path ⑤), resulting in a deadlock. To break this circular dependency and consequently the deadlock, we introduce virtual channels in the rings. By logically separating requests and responses into different virtual channels, we break the chain of dependencies that leads to a deadlock. This logical separation allows the response packets (packet C) to use the physical channel, even if the request-virtual channel is backpressured. Therefore, allowing packets from the Metadata FIFO to exit the HBM-NoC interface module freeing up slots for the request packets without creating circular dependencies.

### 3.7.4 Floorplan

Similar to BFT and Hoplite NoC, we floorplan the Hybrid NoC to span the chip. In Figure 3.17 we show a top-level view of the floorplan used for Hybrid NoC. From the figure, it can be observed that the ring NoC is folded as described in Section 3.5.4 to reduce the path delay on the loop around the link without spending any additional resources in pipelining. Ring and BFT switches along with PEs from the last two rows assigned to SLR0, whereas two rows of BFT and the corresponding Ring NoC switches are assigned to SLR1, and

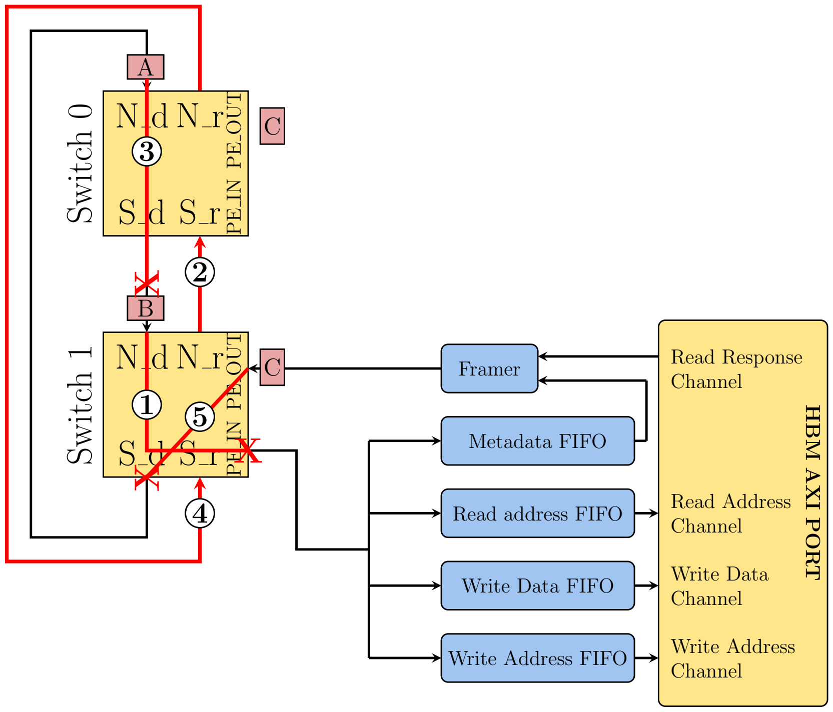


Figure 3.16: Example of a Deadlock in Hybrid NoC. ① Packet B wants to route to the HBM-NoC interface module, but it is blocked because Metadata FIFO is full. ② As a result, the north ready (N_r) of the Switch 1 is pulled down. This blocks packet A wanting to route to the south port at switch 0 ③, and the N_r port of switch 0 is backpressured at the same ④. Which then blocks the packet C from the HBM-NoC module wanting to leave the Metadata FIFO by exiting the through south port ⑤, leading to deadlock

SLR2 gets only one row of BFT. Each link in the network is 261 bits wide, where 256 bits are data bits, and 5 bits are control bits. Therefore, 12528 SLLs are required at each SLR crossing, well within the available SLLs post Vitis consumption. In Figure 3.18 we show the post Place and Route device view of the Hybrid NoC implementation (without PEs) in Alveo U280. In Figure 3.18a we show the leaf cells of NoC and FIFOs in orange, and the leaf cells of other supporting modules instantiated by Vitis are shown in blue, and in Figure 3.18b we show the congestion heat map of the same. We tabulate the resource consumption and the maximum frequency with and without different PEs in the results section.

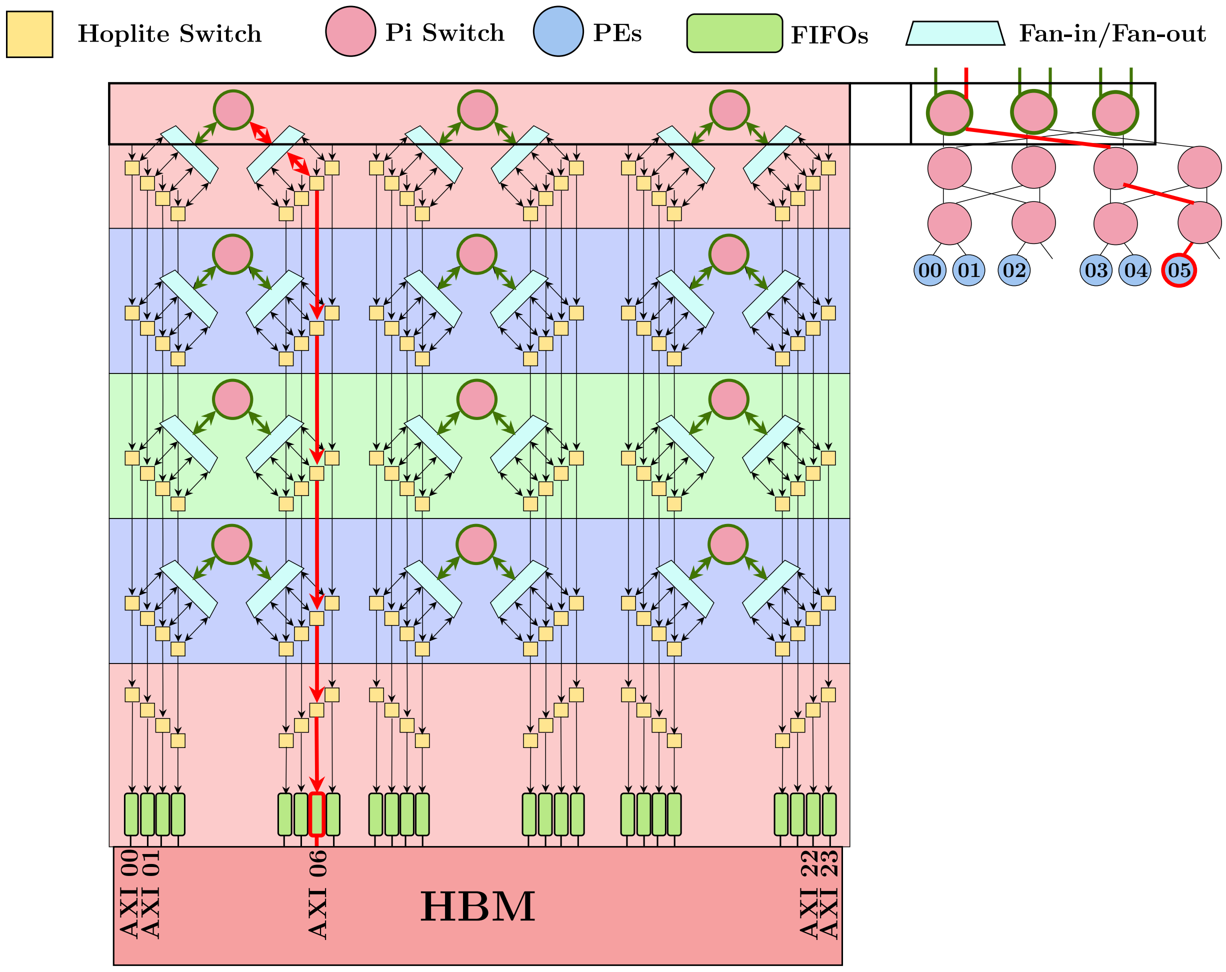
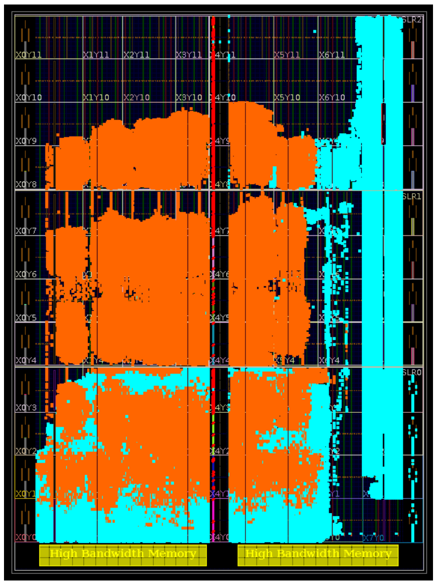
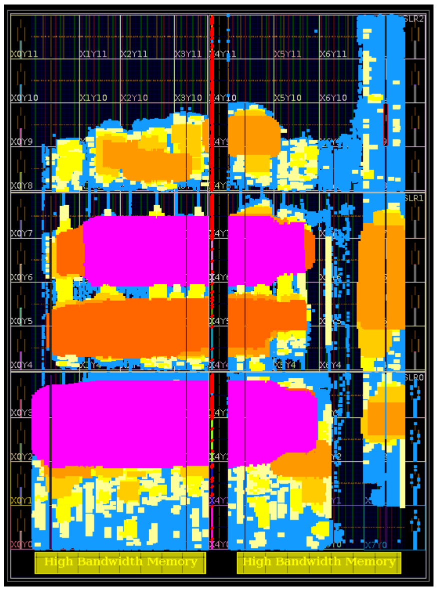


Figure 3.17: PEs interfaced to HBM using Hybrid NoC. Modules in the RED region are assigned to SLR0, including HBM ports. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The path highlighted in red is taken by memory packets injected by PE06 with the memory address set to $60000000h$.



(a) The leaf cells highlighted in orange are instantiated by our design. Whereas, the leaf cells highlighted in blue are from modules instantiated by Vitis.



(b) Congestion Heat Map. The regions highlighted in blue suffer no congestion, and the regions with congestion level 1 are shown in yellow. As the congestion level increases, the colour becomes redder. Similar to BFT the maximum congestion level seen in Hybrid NoC is level 6, shown in pink.

Figure 3.18: Post Placement and Route Device View for Hybrid NoC without PEs.

# Chapter 4

# Benchmarking

In order to measure the effect of our NoCs on the achievable HBM throughput, we benchmark the HBM using synthetic and application-based workloads, where the benchmarks are designed to stress test the internal-cross bar by saturating the HBM ports with reads and writes. The synthetic benchmark helps establish broad trends in throughput variations for different traffic patterns. However, the synthetic benchmarks are not indicative of any actual patterns that the network and HBM could experience with a real application. Therefore, we also designed two application-based benchmarks - Blocked Dense Matrix-Matrix Multipler (DMM) and Sparse Matrix-Vector Multipler (SpMV) - to examine the effects on throughput for a more realistic traffic pattern. In the following sections, we describe in detail the synthetic and application-based benchmarks used in our experiments.

## 4.1 Synthetic Benchmark

Synthetic benchmarks are artificial workloads that are designed to mimic network operations that an actual application could make. They are useful in stress testing the network with different access patterns with the same bitstream. This eliminates the time cost of implementation and bitstream generation, accelerating the initial development phase and quickly establishing the initial performance trends of the system, before developing and implementing time-consuming application-based benchmarks [21]. Consequently, we develop a synthesizable synthetic benchmarking PE in RTL to perform several synthetic micro-benchmarks in hardware, with different memory access patterns controllable at the runtime. They are used to study the impact of the internal-cross bar on the throughput for different access patterns, as listed in Table 4.2, and then the effect of our proposed overlay NoCs on the throughput for the same access patterns.

### 4.1.1 Single Flit

The synthesizable PE tries to saturate the HBM by issuing memory transactions every cycle. First, it writes to the HBM every cycle, where the total number of writes it makes is determined at the compile time. Our experiments set the total writes that a PE makes to 256 MBs, enough to write to every memory location in a pseudo channel. For a single flit network, each transaction is 32B. Therefore, the PE makes $2^{23}$ write transaction. Then, immediately following the writes, the PE issues $2^{23}$ read requests totalling 256 MBs, reading the same memory locations it had previously accessed to write. In total, the PE makes $2^{24}$ memory transactions totalling 512 MB of data.

To verify the functional correctness of the network, i.e no packets are dropped by the network, the PE writes the address of the accessed memory location back to it. For instance, if the PE writes to memory address 700001E00h, the write data is set as 700001E00h. So, when the PE reads back these locations, it verifies if the read data is the same as the address of the memory location. We use the Metadata FIFO in the HBM-NoC interface module to aid this. Along with the source address of the PEs that make the read request, it also stores the read address. This is sent back to PE along with the read data, which the PE then uses to verify if the writes have taken place as intended. After receiving $2^{23}$ read responses from HBM, the PE writes back the total cycle count it took to complete this workload, along with an error bit to HBM.

#### Micro Benchmarks

The PE generates different traffic patterns by varying the memory address. As explained in Section 2.1.1, the 8GB memory space of HBM is split into 32 pseudo channels, each with a capacity of 256 MB. Therefore, 5 bits (32 down to 28) are required to determine the pseudo channel and 28 bits (27 down to 0) to determine the offset of the memory location within the pseudo channel. The 5 bits that determine the pseudo channel is called as **port_id**, and it is encoded in bit 288 down to 284 in a single flit network.

To generate various micro-benchmarks, The PE varies this **port_id** based on an addressing policy, base number (N) and the radius (R). The radius and addressing policy are runtime variables that can be set through the host code and varied for every run. However, the base number (N) is a compile-time variable, and it is fixed for a bitstream. The addressing policy determines the relationship between the base number N and the **port_id** that the PE uses for its memory transactions. Whereas the radius defines the range for **port_id** that the PE is allowed to use. In other words, it determines the range of potential pseudo channels that the PE can use in its memory transactions. Equation 4.1 is used by the PE to get the **port_id** for a base number N with radius R. It generates R possible values for **port_id** around N. For instance, if R is 4, the possible values of $getPortId(N, 4)$ is $N + 1, N, N - 1$, and $N - 2$.

$$getPortId(N, R) \ = \ N \ + \ (randInteger() \mod R) \ - \ (R/2)) \tag{4.1}$$

For every memory transaction, the PE, depending on the addressing policy, could vary the target **port_id**. When the radius is set to 1, $(randInteger() \mod R)$ becomes zero. Therefore only one value for **port_id** is possible. However, as the radius increases, the number of possible values for **port_id** increases, and the PEs randomly choose a new one for every memory transaction. To aid it in generating a random number for $getPortId()$, we use a 16 bit LFSR with only the least significant eights bits used as the $randInteger()$. Since the read must use the same memory locations as the writes, the LFSR is progressed only when a new memory transaction is initiated, and at the end of phase 1 (write phase), the LFSR is reseeded. Therefore, it repeats the same sequence for phase 2 (read phase). The offset determines the memory location within a pseudo channel, and the PEs are set to generate them linearly.

We use seven addressing policies - Peer to Peer (P2P), Cross-Bank (CB), Cross-Stack (CS), Nearest-Neighbour (NN), Criss-Cross (CC), Tornado (TO), and Bit-Reversal (BR) - in our experiments, and by combing some of the addressing policies with a radius factor, we generate a diverse set of micro-benchmarks. The Table 4.2 tabulates the formulas used by the PEs to generate the target **port_id** for its memory transactions for various addressing policies. For Cross-Bank addressing policy, each PE will access pseudo channels in the next bank. When the radius is 1, only one pseudo channel is accessed by the PE. With the radius set to 4, PE will access all pseudo channels of the next bank, taking care to avoid Cross-Stack access for underflow and overflow conditions. Similarly, for the Cross-Stack addressing policy, each PE will access pseudo channels in the next stack. When the radius is 1, only one pseudo channel is accessed by the PE. With the radius set to 16, PE will access all pseudo channels of the next stack. In Figure 4.1 we show the pseudo channel(s) that will be accessed by PE with base number 16 for some of the addressing policies. This figure assumes that a PE with base number N is interfaced to HBM AXI port N. The post-placement and routing resource consumption and the maximum operating frequency of the Synthetic Benchmarking PE is tabulated in Table 4.1

| **LUTs** | **FFs** | **Frequency** |
|---|---|---|
| 778 | 1055 | 394 MHz |

Table 4.1: Post Place and Route resource utilization of a Synthetic Benchmarking PE. The frequency reported here is the maximum frequency observed for a Synthetic Benchmarking PE with noNoC design.

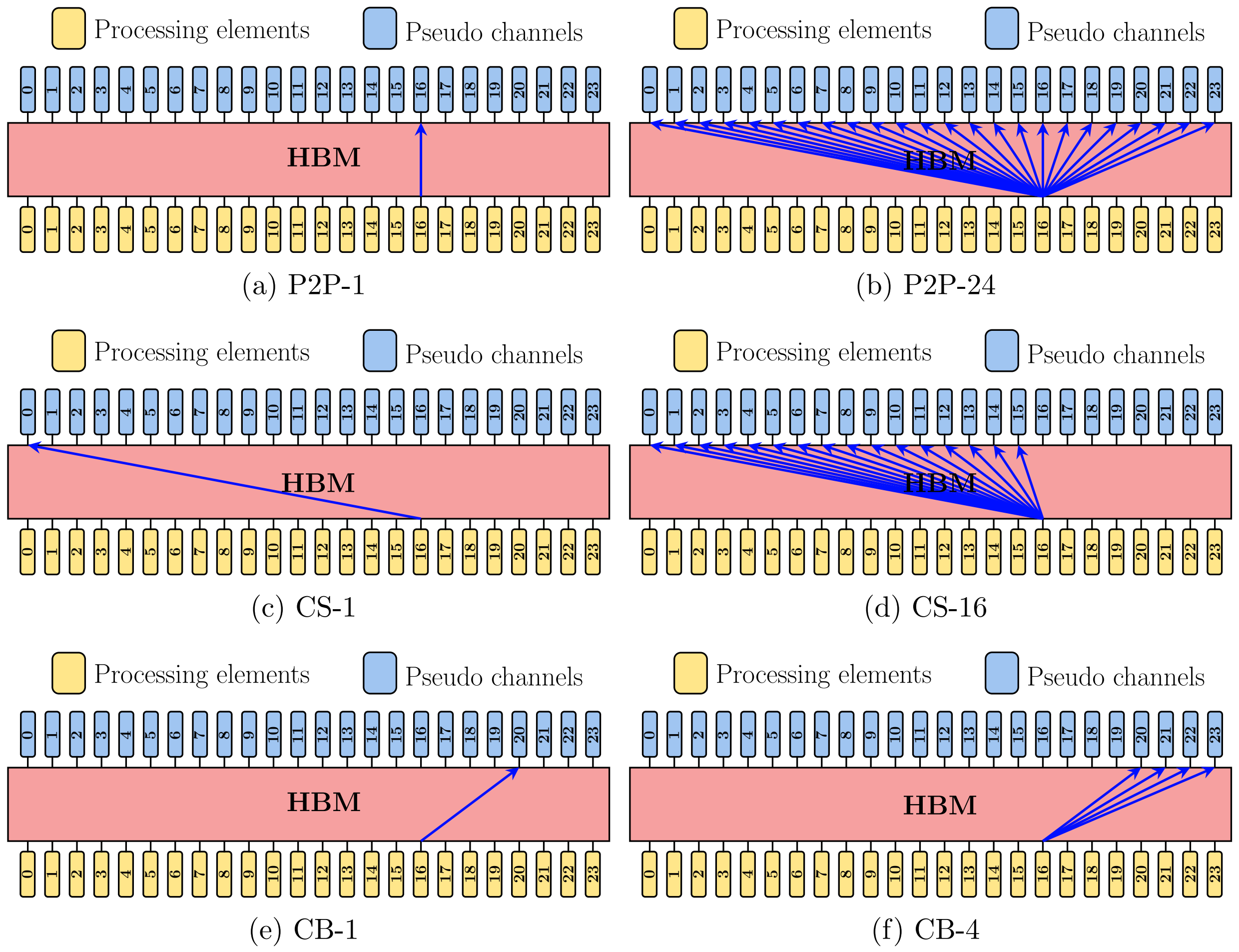
| **Addressing Policy** | **Target port_id (P)** |
|---|---|
| Peer-to-Peer (P2P) | $P = getPortId(N, R)\%24$ |
| Cross-Bank (CB) | $P = getPortId(N + 4, R)\%24$ |
| Cross-Stack (CS) | $P = getPortId(N + 16, R)\%24$ |
| Nearest-Neighbour (NN) | $P = (N + 1)\%24$ |
| Criss-Cross (CC) | $P = 24 - N$ |
| Tornado (TO) | $P = (N + 12)\%24$ |
| Bit-Reversal (BR) | $P = bitreversal(N)\%24$ |

Table 4.2: Addressing Policies used for Synthetic Benchmarks

### 4.1.2 Multi-Flit

The multi-flit synthetic benchmarking PE is similar to the single-flit PE, barring a few differences. In the single flit PE, the number of transactions it makes is fixed to $2^{23}$ per phase. However, for the multi-flit PE, the number of transactions required to cover the same memory size of 256MB depends on the burst length, and it is determined by equation 4.2. Furthermore, along with the radius and addressing policy, for multi-flit PEs, the burst length can also be varied at the runtime through the host code.

$$totalTransactions = 2^{23-log_2(burst\_length)} \tag{4.2}$$



(a) P2P-1 (b) P2P-24

(c) CS-1 (d) CS-16

(e) CB-1 (f) CB-4

Figure 4.1: Pseudo channel accessed by PE with base number 16 for various addressing policies, assuming that PEs with base number N is interfaced to HBM AXI port N

## 4.2 Application Benchmark

While the synthetic benchmarks help accelerate the initial development phase and quickly establish initial network performance trends, they are not indicative of the patterns that the network might experience with real applications. For example, the synthetic benchmarks cannot capture the complex memory access patterns, data dependencies and varying injection rates. To that end, we design two application-based benchmarks, Sparse Matrix-Vector Multiplier and Blocked Dense Matrix Matrix Multiplier, in HLS.

### 4.2.1 Blocked Dense Matrix Multiplication

We multiply two dense, square matrices, A and B, to compute matrix D in a blocked manner. The output matrix D is broken down into multiple smaller matrices called blocks, as shown in Figure 4.2, and they are computed in parallel by different PEs. By assigning PEs to compute different output blocks in parallel, we exploit the fine-grain parallelism of Matrix-Matrix multiplication. The processing element consists of two sub-modules, a scheduler and compute module, and it is interfaced to the NoC through two AXIS ports, as shown in Figure 4.3a. One AXIS port acts as the master, and it is used to initiate transactions that write to the network, such as write and read request transactions. The other AXIS port acts as the slave, and it is used to read from the network. The scheduler module orchestrates the movement of data to and from the HBM. It calculates the required number of memory transactions and the addresses based on the dimensions of the input matrices and then issues read-requests and write transactions. The compute module is comprised of DSP blocks, on-chip URAM memory, and a register file, which are used to perform the multiply and accumulate operation.

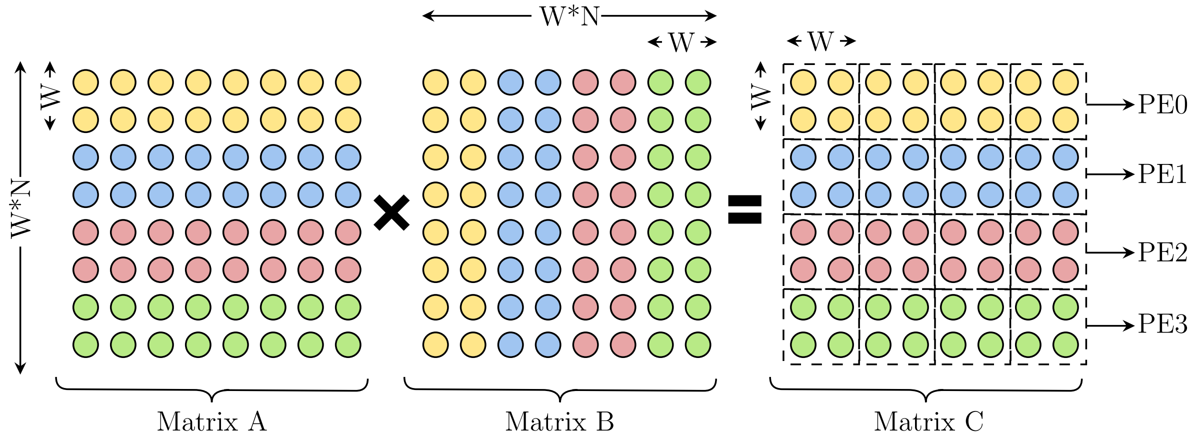
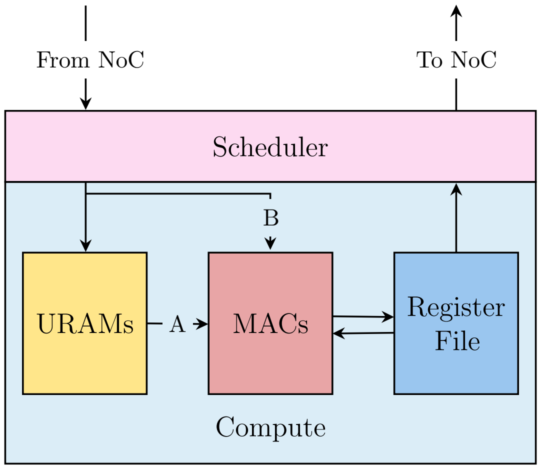
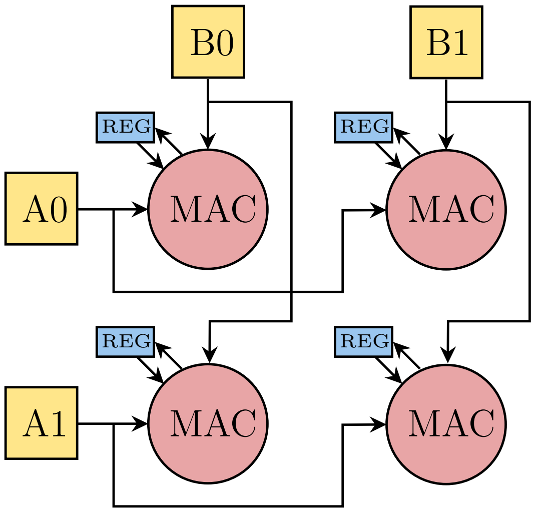


Figure 4.2: Example of a blocked dense matrix-matrix multiplication, where the output block size W is set to 2, and the number of PEs is 4. Each PE is assigned a row of output blocks, and to minimize the crowding at the AXI ports, the elements of Matrix A, B and C are split across multiple pseudo channels. The elements with the same colour are placed in the same pseudo channel, and each colour is assigned a different pseudo channel

In our implementation, we have restricted the input matrices to square matrices with dimensions divisible by $W \times N$, where $W$ is the dimension of the output blocks, and $N$ is the number of PEs used. First, consider an example where the input matrices dimension is



(a) Components of Blocked Dense Matrix Matrix Multiplication PE.



(b) MACs in the compute module (W=2).

Figure 4.3: Blocked Dense Matrix Matirx Multiplication PE

equal to $(W*N)\times(W*N)$. In Figure 4.2, we show an example of this, where the dimension of the output block is 2, and the number of PEs used is 4. Therefore, the dimension of the input and the output matrices is 8×8. Each PE is assigned to compute N output blocks (four in the example), all from the same row. This assignment allows the PEs to store and reuse rows from matrix A, minimizing expensive memory accesses. The PE operates in three separate phases. In Phase 1, the PE retrieves $W$ rows in matrix A, stores them internally in the URAM. In Phase 2, the PE then retrieves $W$ columns from matrix B. As the elements of matrix B are read from memory, they are multiplied with the stored row elements of matrix A in the fly, as shown in Figure 4.3b. The PE performs $W^2$ multiply and accumulate operation in a clock cycle. Finally, in Phase 3, after the required number of MAC operations have been completed, the computed output block is written back to the HBM. Since each PE is assigned $N$ output blocks to be computed, the PE repeats Phase 2 and Phase 3 $N$ times, computing $(W) \times (W * N)$ elements of the output matrix. So, with $N$ PEs, we can compute the $(W * N) \times (W * N)$ elements of the output matrix in parallel.

Since the PEs performs $W^2$ MAC operations in parallel, any increase in the output block dimension $W$ will result in a quadratic increase in the number of MAC modules, implemented using DSPs, required per PE. This puts an upper limit on the output block dimension $W$. So, for an input matrix dimension significantly larger than $(W*N)\times(W*N)$, we need to break the output matrix down into smaller patches of size $(W * N) \times (W * N)$, and compute the patches one at a time. For instance, consider the example shown in Figure 4.4. Similar to the previous example, the dimension of the output block is 2, and the number of PEs used is 4. But, the input matrix size is 16 instead of 8. Therefore, the output $16 \times 16$ matrix is broken down into four patches, each with a dimension of $8 \times 8$, and as explained above, the PEs now computes one patch at a time. To minimize the number of memory transactions, the PE reuses the stored elements of matrix A by computing patch 0 followed by patch 1, then patch 2, and finally patch 3. Since all the PEs operate in parallel, care must be taken to avoid crowding at the ports, where all the PEs access the same AXI port. To minimize this, the inputs A and B are split across several pseudo channels, and the PEs compute the output blocks in a staggered manner.

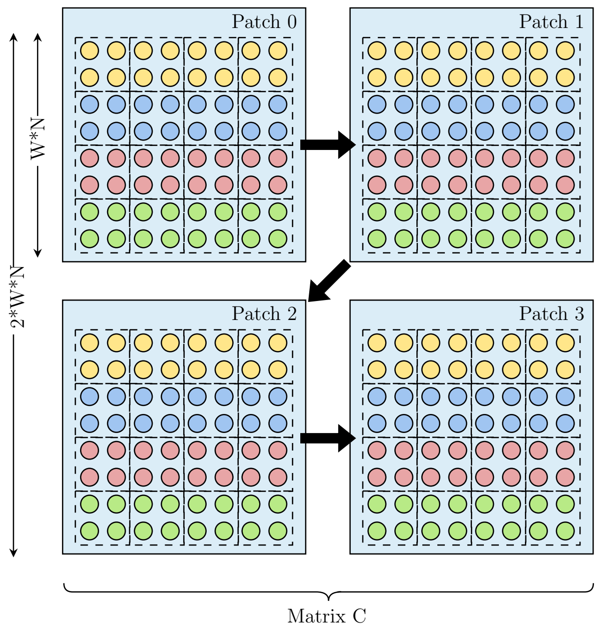


Figure 4.4: Example of a blocked dense matrix-matrix multiplication where the input matrix size is $(2 * W * N) \times (2 * W * N)$. The output block size W is set to 2, and the number of PEs is 4. The larger output matrix is broken down into smaller patches of dimensions $(W * N) \times (W * N)$, and they are in series.

We designed this PE using HLS, and they are optimized for throughput. The sub-functions of the PE are fully pipelined and are spatially unrolled when possible. Similarly, we ensure overlay between communication and computation, when possible. However, due to control flow requirements for dataflow pragma, the design as a whole is not fully pipelined. Based on the HLS reported numbers, for a matrix size of 1024 and a block size of 8, a PE takes 601092 cycles to complete its execution, where 524800 (87%) cycles are compute cycles. This calculation assumes that the required data are available immediately at the ports. The matrix dimension of the inputs is configurable at the runtime, and the PE can support a maximum matrix size of 20480. Considering the resource utilization, the width of the AXIS port of the PE and the HBM, the output block width is set as 8, and it is not configurable. The design fails to complete placement and routing with 24 PEs. Therefore, we reduce the number of PEs to 16. The post Place and Route resource utilization for a PE are tabulated in Table 4.3.

| **LUTs** | **FFs** | **DSPs** | **URAMs** | ***Frequency** |
|---|---|---|---|---|
| 5729 | 4677 | 192 | 20 | 320 MHz |

Table 4.3: Post Place and Route resource utilisation of a blocked dense matrix matrix multiplication PE. *Standalone operating frequency reported by HLS.

### 4.2.2 Sparse Matrix Vector Multiplication

The second application-based benchmark that we designed was a Sparse Matrix-Vector Multiplication, where we multiply a sparse matrix $\mathbf{A}$ of size $M \times M$ and with a vector $\mathbf{x}$ of size $M$ resulting in an output vector $\mathbf{b}$ of size $M$. In our implementation, the output vector $\mathbf{b}$ is reused as the input vector $\mathbf{x}$ for every new iteration of the multiplication. The number of times this process repeats is dictated at the runtime via the host code. Similar to the DMM, the workload is evenly divided across several PEs, where each PE is assigned to compute some of the output vector elements, as shown in Figure 4.6. Furthermore, the skeletal structure of the SpMV PE is similar to the DMM PE. Like DMM PE, the SpMV PE also consists of two sub-modules, a scheduler and compute module, and it is interfaced to the NoC through two AXIS ports, as shown in Figure 4.5. One AXIS port acts as the master, and the other AXIS port acts as the slave. The scheduler module orchestrates the movement of data to and from the HBM. It calculates the required number of memory transactions and addresses based on the input matrix A's sparsity, then issues read-requests and write transactions accordingly. The compute module is comprised of DSP blocks and on-chip URAM and BRAM memory, which are used to perform the dot product between the sparse matrix and the vector.

Since matrix A is being resued between iterations, it is read from memory once and stored in the PE's memory. Therefore, we use the Compressed Sparse Row (CSR) format

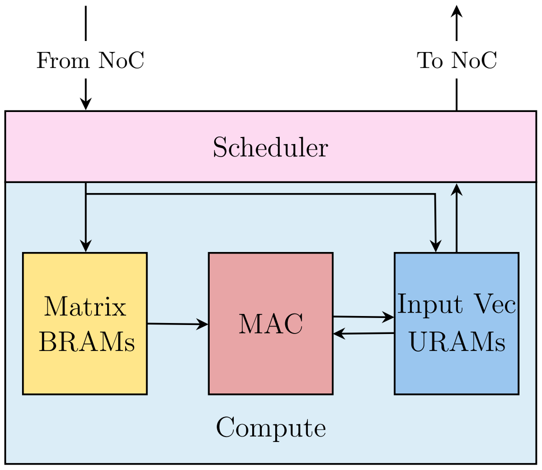


Figure 4.5: Sparse Matrix Vector Multiplication PE

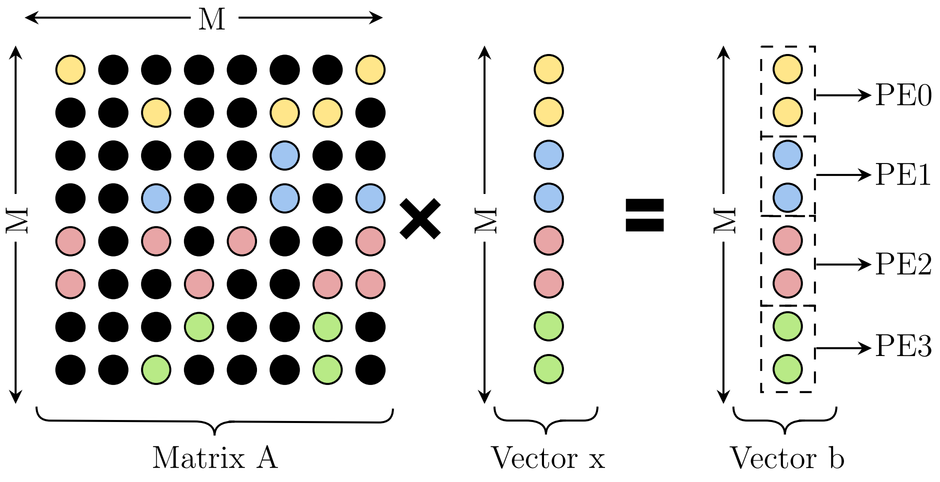


Figure 4.6: Example of a Sparse matrix-vector multiplication where the input matrix size $M$ is 8×8, and the number of PE is 4.

to encode the sparse matrix **A**. A matrix with $Z$ non-zero values and a dimension $M$ requires $2 * Z + M$ memory locations. Other formats such as Coordinate Format (COO) would require $3 * Z$ memory locations. The choice of CSR as the encoding format also eliminates the possibility of partitioning the input vector and storing them in different buffers. Without the complete row coordinate information (as available in COO), it is impossible to multiply the streamed input vector values as they come in with an initiation interval of 1. The PE needs to loop through the encoded CSR sparse matrix **A** to identify the matrix elements and multiply the streamed in vector elements in the fly. Similarly, unrolling the multiplication to create an adder tree is also infeasible, as it would require multiple copies of the matrix, increasing the memory requirements. So each PE needs to store the entire input vector **x**, and the multiplication is pipelined with an initiation interval of 1 instead of unrolling.

The PE operates in four phases. In the first phase, the PE reads and stores the required rows of the sparse input matrix **A** and the input vector **x**. Then in the second phase, it multiples the stored rows of matrix **A** and the input vector **x**, and after the end of the second phase, each PE would have computed the elements of the output vector **b** assigned to it. Since the output vector **b** is used as the input vector to the next iteration, in the third phase, the PE broadcasts the output vector elements that it computed, and parallelly it stores the output vector elements broadcasted by other PEs. Then based on the iteration count set at the runtime, the second and third phase is repeated several times. Finally, after multiplying for the set number of times, in the fourth phase, the PEs write back the output vector **b** to HBM. In the presence of an NoC, the PEs use the network to exchange the computed output vector elements. If an NoC is not present, the HBM is used instead. Like DMM, care is taken to avoid crowding at the ports of the PEs during the third phase.

| **LUTs** | **FFs** | **DSPs** | **URAMs** | **BRAMs** | ***Frequency** |
|---|---|---|---|---|---|
| 5608 | 3651 | 9 | 8 | 32 | 390 MHz |

Table 4.4: Post Place and Route resource utilisation of a Sparse Matrix-Vector PE. *Standalone operating frequency reported by HLS.

| **Benchmark** | **Dimension** | **No. Non-Zeros** |
|---|---|---|
| dw8192 | 8,192 | 41,746 |
| epb | 14,734 | 95,053 |
| t2d_q9 | 9,801 | 87,025 |

Table 4.5: Sparse Matrix Benchmarks used as input workloads in our experiments.

Since the workload needs to be divided equally between all PEs, the dimension $M$ of the input vector and matrix must be a multiple of $N$, where $N$ is the number of PEs in the system. If the dimension input matrix is not a multiple of $N$, it must be padded with zeros to make it a multiple of $N$, and the CSR encoding must be adjusted accordingly. The PEs can accept a maximum matrix size of up to 16368, set during runtime. The input matrix can have up to 98304 total non-zero values, provided each PE is not assigned more than 4096 non-zero values. Like the DMM PE, the SpMV PE is optimized for throughput, where every sub-function is fully pipelined and completely unrolled when possible. For example, with an input matrix of dimension 1024*1024, with 65536 non-zero values, and the iteration count set to 16, the PE takes 46289 cycles to complete, wherein 34935(75%) cycles are compute cycles. This calculation assumes that the required data are available immediately at the ports. The post Place and Route resource utilization for a PE are tabulated in Table 4.4, and in Table 4.5 we show the dimensions and the number of non-zeros in various input sparse matrix benchmarks used in our experiments.

# Chapter 5

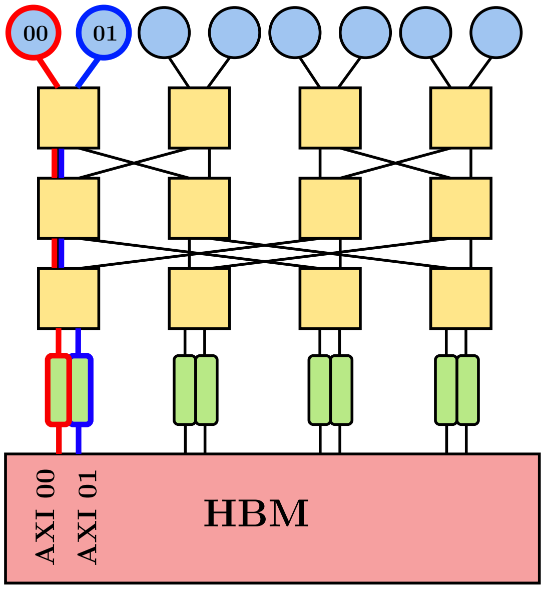
# Evaluation

This chapter first covers the evaluation and hardware setup used in our synthetic and application-based benchmarking experiments. Then we report the maximum operating frequencies of various designs used in our experiments, followed by the results of synthetic benchmarking. Finally, we report the results of the Sparse Matrix-Vector Multiplication and the Dense Matrix-Matrix Multiplication benchmarks.

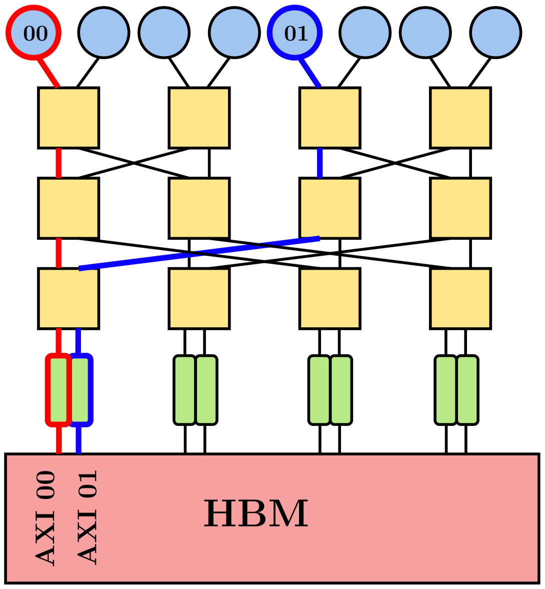
## 5.1 Evaluation Setup

The goals of our experiments are to characterize the existing HBM throughput bottleneck and evaluate the impact of NoC overlays on it using synthetic and application benchmarks. To that end, we designed a synthesizable synthetic benchmarking PE, and two application-based benchmarks, Dense Matrix-Matrix Multiplication and Sparse Matrix-Vector Multiplication PEs. To establish a baseline to compare the effect of the NoCs on throughput, we first interface the benchmarking PEs directly to the HBM ports; we call this the noNoC design. Then we repeat the same sweep of experiments with PEs interfaced to HBM using our NoCs. Since the NoCs are interfaced to the HBM ports through the FIFOs of the HBM-NoC interface module, the presence of FIFOs offers a boost to the attained throughput for PEs with the NoCs. Therefore, even for noNoC designs, we use the HBM-NoC interface module to keep comparisons fair.

The HBM port to which a PE is connected affects the achievable throughput, and the best choice for these connections that maximize throughput depends on the workload. In our experiments, we choose the configuration of PEs within and without the NoC that maximizes the achievable throughput for P2P addressing policy with a radius set to 1. We choose this addressing policy since it does not require the lateral links of the internal crossbar, and therefore the throughput remains unaffected by lateral links. For instance, a PE with base number 2, if directly connected to AXI port 16, will have poorer throughput performance with P2P-1 addressing policy than if it was connected to AXI port 2, instead. We do the same for PEs interfaced to the HBM using NoCs. The relative position of the PEs within a network is chosen to minimize the network contention for P2P-1 addressing policy, thereby improving the achievable HBM throughput. For instance, in Figure 5.1 we show an example of PE arrangement within BFT with eight PEs. If PEs with base numbers 0 and 1 are arranged shown in Figure 5.1a, they contented for the same uphill links for P2P-1 addressing policy. Instead, if arranged as shown in 5.1b, they use different uphill links and can ascend the network without contention.



(a) PE00 and PE01 share the same link for P2P-1



(b) With change in arrangement PE00 and PE01 use different links for P2P-1

Figure 5.1: Example arrangement of PEs within a BFT network.

### 5.1.1 Hardware Setup

We use the Vitis 2020.2 toolchain to compile the RTL implementation of our designs for the Xilinx Alveo U280 datacentre FPGA. To do this, we package our design as a kernel with 24 top-level AXI ports, and we use Vitis configuration files to map the kernel ports to HBM AXI3 ports. As explained in Section 2.4.1, Vitis instantiates and configures PCIe interfaces for communication between the host and the target platform (Figure 2.12), inserts AXI switches between the kernel and HBM AXI ports, and configures the HBM subsystem IP. Apart from assigning the mapping between the kernel ports and HBM AXI ports, we do not make any modifications to the Vitis instantiated modules, and they are left with default configurations.

We use an OpenCL-based host code to load the HBM with required values, execute the kernel, retrieve the result from HBM, and verify the correctness. For SpMV and DMM, we execute the PEs with different workloads, and through host code, the functional correctness has been verified for all the workloads used. Similarly, we also use Vitis to measure the HBM throughput and the runtime of our kernels, by enabling the profiling counters. These additions have a small impact on the operating frequency but are necessary for visibility into the performance.

However, Vitis does not account for the entire runtime of the kernel. Instead, it considers only the time the ports were active in responding to the memory transactions of the kernel. In other words, Vitis only considers the time between the start of the memory transactions and the end of transactions in its throughput calculation. This is acceptable for synthetic workload, where the port is active throughout the kernel execution. However, for application-based benchmarks, such as SpMV and DMM, where the ports could remain idle for a portion of the kernel's run time, the results from Vitis indicate throughput per port rather than the throughput per PE. Therefore, for application-based workloads, along with the throughput reported from Vitis, we also report the runtime.

## 5.2 Operating Frequencies

In Table 5.1, we tabulate the maximum operating frequencies of our designs. The frequencies given in Table 5.1a are for the single-flit designs, whereas frequencies in Table 5.1b are for the multi-flit designs. In our subsequent discussions, to differentiate single-flit and multi-flit NoCs, we use '-SF' for single-flit and '-MF' for multi-flit.

Without any PE, i.e., with just the network, the Hoplite has the highest operating frequency of 340 MHz. This can be attributed to the simple torus topology of the Hoplite, which can be mapped to a 2D chip better than the BFT or Hybrid. Both BFT and Hybrid suffer from hierarchical topology requiring highly nested floorplanning that is unnecessary in Hoplite. As explained in Section 3.6.4, this leads to higher congestion and lower fre-

| NoC Type | No PE | Synthetic PE |
|---|---|---|
| **Hoplite** | 340 MHz | 306 MHz |
| **BFT** | 327 MHz | 300 MHz |
| **noNoC** | - | 340 MHz |

(a) Operating Frequencies for Single-Flit designs

| NoC Type | No PE | Synthetic PE | DMM PE | SpMV PE |
|---|---|---|---|---|
| **BFT** | 301 MHz | 272 MHz | 251 MHz | 203 MHz |
| **Hybrid** | 297 MHz | 290 MHz | 256 MHz | 236 MHz |
| **noNoC** | - | 300 MHz | 254 MHz | 174 MHz |

(b) Operating Frequencies for Multi-Flit designs

Table 5.1: Maximum Operating Frequencies for designs used in our experiment

quency. However, the Hoplite does not support multi-flit PEs such as DMM and SpMV, and it suffers from the livelock, limiting its performance.

The BFT-MF without any PEs has a lower operating frequency than the BFT-SF. This can be attributed to the lock system that has been incorporated in the switches of BFT-MF, which increases the number of LUT levels in the critical path, thereby reducing the operating frequency. Moreover, the BFT-MF without PEs operating frequency is marginally higher than the Hybrid-MF. However, with any PE - synthetic, DMM or SpMV - the operating frequency of BFT-MF drops below the Hybrid-MF. Again, this can be attributed to the highly nested floorplanning used for the BFT. While the Hybrid also has some nested floorplanning for its BFT rows, it is only three levels of nesting. With BFT, we need five levels of nesting, worsening the congestion when PEs with high resource utilization such as SpMV is interfaced with the network.

Synthetic benchmarking PEs consumes fewer resources than SpMV or DMM PE. Therefore, they have a higher operating frequency than their counterparts with an NoC. But for SpMV PE, without any network, the Vivado struggles to place and route the PEs in SLRs other than SLR0. Consequently, the SpMV PE operates at a lower frequency when compared to SpMV with an NoC. We use 24 PEs in our SpMV experiments. In comparision, only 16 PEs are used in DMM experiments. As a results, without any floorplan directives, the noNoC design experiences a larger drop in frequency with SpMV PEs.

To summarise, for PEs with significant resource consumption such as SpMV and DMM PEs, the Hybrid NoC offers the highest operating frequency.

## 5.3 Synthetic Benchmarking

In this section, we present the results from our Synthetic benchmarking experiments. We first present the effect of global accesses on throughput and show the throughput gains offered by our NoCs. Then, we do the same for cross-stack, cross-bank, and other access patterns used in our experiments.

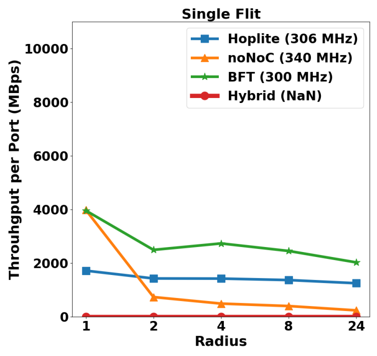
### 5.3.1 Effect of Global accesses

In Figure 5.2, we plot the throughput per port against P2P addressing policy with different radii, and different burst lengths. As the radii increase, the number of pseudo channels that a PE could access increases, and with a radius of 24, a PE can access all the 24 pseudo channels used in the system, mimicking global accesses. On the other hand, with the radius set to 1, the PEs access only one pseudo channel, and the lateral links are un-used. The operating frequency determines the achievable throughput for radius 1, and since the synthetic benchmarks without an NoC operate at a higher frequency, they achieve higher throughput than PEs with an NoC. As the radii increase, the achievable throughput without NoCs quickly drops off, despite the higher operating frequency.

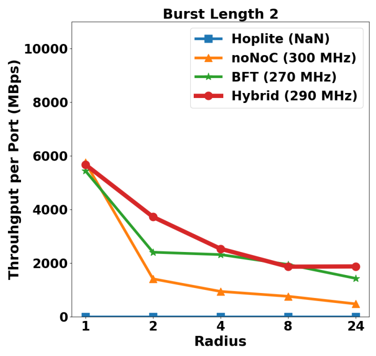
Similarly, with the increase in burst length, going from single-flit to burst length 16, the effectiveness of each memory transaction improves. For single-flits, the maximum throughput utilization for any design is 43%, delivered by BFT. With the increase in burst length, the utilization increases, achieving a maximum utilization of 98%, delivered by the noNoC multi-flit design with burst length set to 16.

With the single-flit PEs, both the NoC design improve the attained throughput. Hoplite NoC, despite its limitations, beats the noNoC design. For full global accesses (Radius 24), Hoplite improves the throughput by five times. However, the BFT offers the highest throughput, for all radii larger than 1. Notably, with the radius set to 24, BFT improves the throughput by a factor of 8.6 compared to the noNoC design.

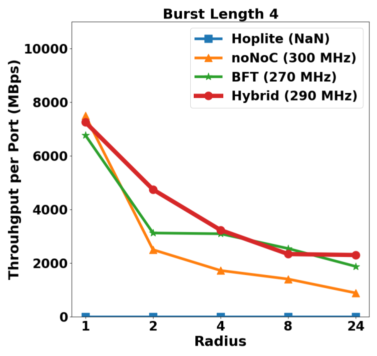
Similarly, for burst lengths 2, 4, and 8, the attained throughput per port is higher with our NoCs. Hybrid offers the highest throughput per port for all data points barring P2P-8



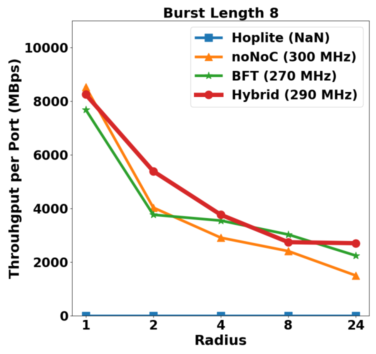
(a) Single-Flit



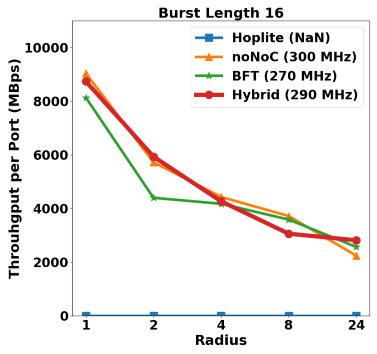
(b) Burst Length 2



(c) Burst Length 4



(d) Burst Length 8



(e) Burst Length 16

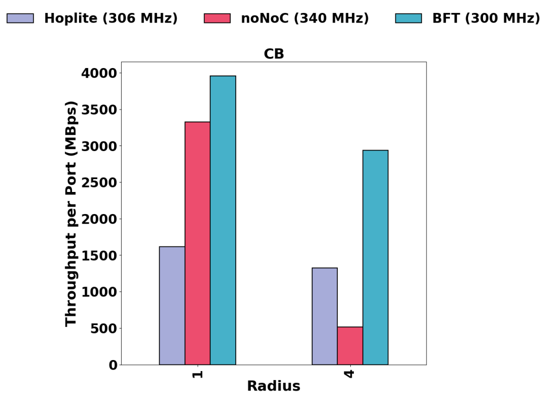
Figure 5.2: Throughput per port versus P2P addressing policy for different burst lengths

with burst lengths 4 and 8, where BFT beats the Hybrid NoC. However, the results are a mixed bag for burst length 16. The BFT loses to noNoC design for all radii, barring 24. Whereas the Hybrid NoC offers higher throughput than noNoC design only for radii 2 and 24, with a speedup of 1.26 for radius 24.

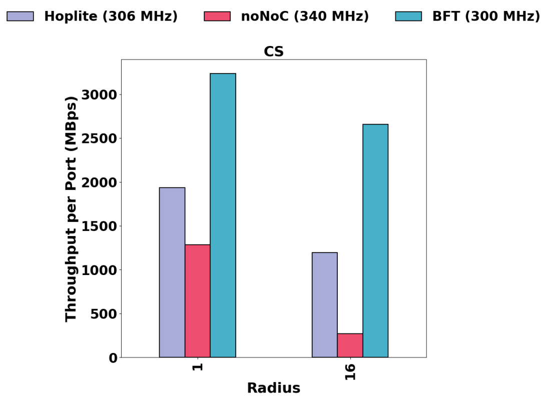
To summarise, our NoCs improve the achievable throughput for global accesses (radius 24) for all burst lengths. However, with the increase in burst length, the effectiveness of our NoCs decreases. For the single-flit PEs, the BFT offers the highest throughput speedup of 8.6. However, for multi-flit PEs with burst length set to 16, the highest throughput speedup is 1.26, delivered by the Hybrid NoC. Barring burst length 16, the designs with NoC offer better throughput than the noNoC designs for other burst lengths. Based on our experiments, BFT emerges as the best choice for single-flit PEs and Hybrid for multi-flit PEs.

## 5.3.2 Effect of Cross-Bank and Cross-Stack accesses

In Figure 5.3a we plot the throughput per port against the radius for single-flit designs with cross-bank addressing policy. Similarly, we plot the throughput versus burst length for multi-flit design with cross-bank addressing policy in Figure 5.4. With Cross Bank addressing policy, each PE accesses the pseudo channel in the adjacent bank, as shown in Figure 4.1. When the radius is set to 1, PEs access only one pseudo channel in the next bank. If the radius is to 4, PEs can access all the pseudo channels in the next bank with an equal probability.



(a) Throughput vs Radius for single-flit designs with Cross Bank addressing policy

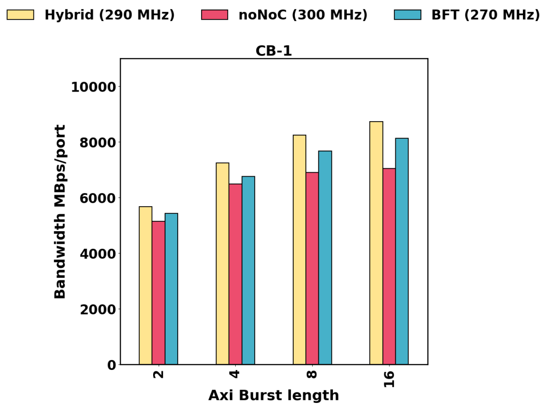


(b) Throughput vs Radius for single-flit designs with Cross stack addressing policy

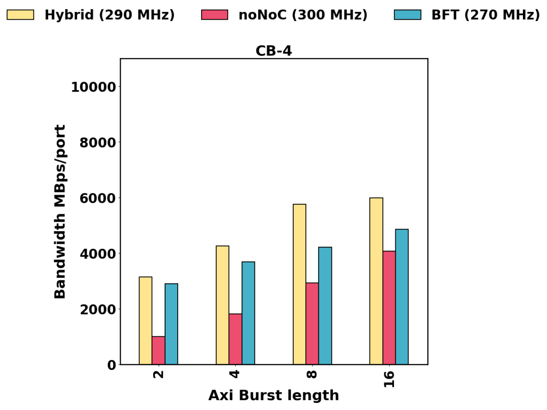
Figure 5.3: Throughput results for single-flit networks with Cross bank and Cross stack addressing policy

From Figure 5.3a, it can be observed that across all data points, the BFT-SF NoC outperforms the noNoC-SF design. Similar to the P2P addressing policy, the throughput attained by noNoC-SF design plummets by 6 times when the radius is increased to 4, and BFT-SF improves this throughput by 5.6 times. For Hoplite-SF, the results are mixed. Imposing limits on pending read requests to eliminate the possibility of livelock, as explained in 3.5.2, degrades the achievable throughput, and consequently, Hoplite-SF performs poorer than the noNoC-SF. Nevertheless, despite this limitation, Hoplite-SF improves the throughput by 2.5 times when compared to noNoC-SF for radius 4.

Both the BFT-MF and Hybrid-MF NoCs surpass the throughput attained by the noNoC-MF design, where the Hybrid-MF outperforms both the BFT-MF and noNoC-MF design, as seen in Figure 5.4. For Cross-Bank addressing policy with radius 2, Hybrid



(a) Throughput vs Burst Length for multi-flit designs with Cross Bank addressing policy with Radius 1



(b) Throughput vs Burst Length for multi-flit designs with Cross Bank addressing policy with Radius 4

Figure 5.4: Throughput results for multi-flit networks with Cross bank addressing policy

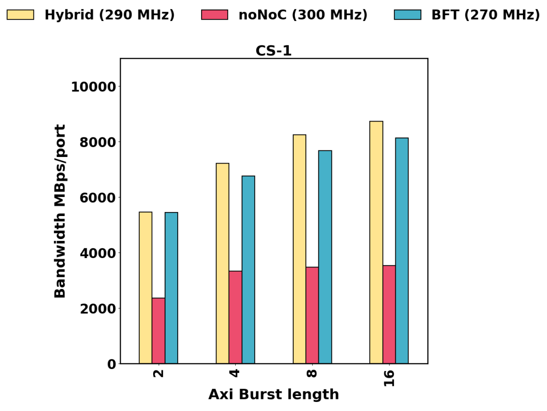
NoC improves the throughput by a factor of 3.14 times (burst length 2). However, similar to the trend observed in P2P addressing policy, the gulf between noNoC-MF and the NoCs narrows with increased burst length. As a result, the throughput speedup delivered by the Hybrid NoC drops from 3.14 for burst length 2 to 1.46 for burst length 16.

We observe a similar trend for Cross-Stack addressing policy as well, as seen in figures 5.3b, and 5.5. In this addressing policy, the PEs access pseudo channels in the other stack, as shown in Figure 4.1. If the radius is set to 1, PEs access only one pseudo channel in the other stack, and the PEs can access all the pseudo channels in the other stack if the radius is increased to 16.

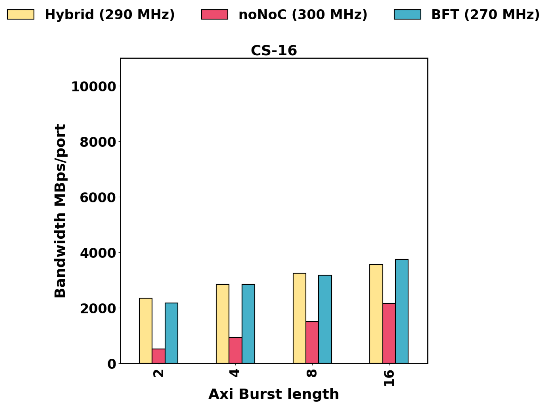
In both the single-flit and multi-flit design, the throughput attained by noNoC-SF drops by a factor of 0.3 times for Cross-Stack accesses compared to P2P-1. This result agrees with the data provided by Xilinx [47]. For single-flit designs (Fig. 5.3b), both the Hoplite-SF and BFT-SF deliver higher throughput than noNoC-SF, with BFT-SF emerging as a clear winner. Unlike the Cross-Bank addressing policy, Hoplite-SF outperforms the noNoC-SF, despite the limit on pending read requests. With radius 1, Hoplite-SF improves the throughput by 1.5 times, and the BFT-SF improves it by a factor of 2.5. With radius 16, the throughput gains are higher. Hoplite-SF improves the throughput by 4.4 times, and the BFT-SF improves it by a factor of 9.8.

Likewise, for multi-flit designs (Fig. 5.5) use of NoCs improve the attained throughput. Unlike Cross-Bank policy, there is no clear winner between the NoCs. Hybrid-MF NoC marginally outperforms BFT for burst lengths up to 8, whereas BFT-MF wins for burst length 16. With burst length 2, Hybrid-MF improves the throughput by 4.4 times, and when the burst length is increased to 16, the throughput speedup drops to 1.7.

In summary, our NoCs improve the throughput achieved compared to the noNoC designs across the boards for cross-stack and cross-bank accesses. Notably, for CS-16, BFT-SF delivers a throughput speed up of 9.8, whereas Hybrid-MF improves the throughput by 4.4 times with burst length two for the same addressing policy. However, similar to the P2P addressing policy, the throughput gains declined with the increase in burst lengths. The highest throughput speedup for CS-16 with burst length 16 was 1.7, delivered by Hybrid-MF.



(a) Throughput vs Burst Length for multi-flit designs with Cross Stack addressing policy with Radius 1



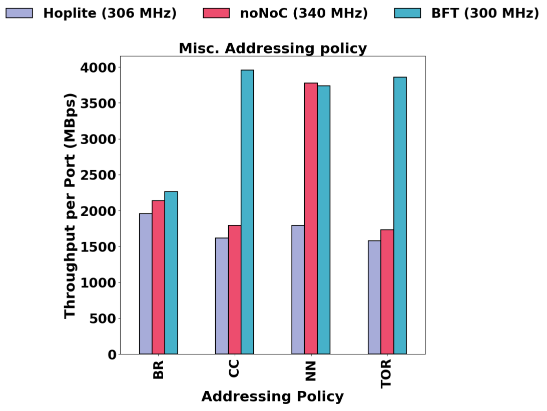
(b) Throughput vs Burst Length for multi-flit designs with Cross Stack addressing policy with Radius 16

Figure 5.5: Throughput results for multi-flit networks with Cross Stack addressing policy

### Other Micro-Benchmarks

The throughput results for Bit-Reversal (BR), Criss-Cross (CC), Nearest-Neighbour (NN), and Tornado (TOR) addressing policies, as defined in Section 4.1.1, are plotted in figures 5.6a and 5.7.

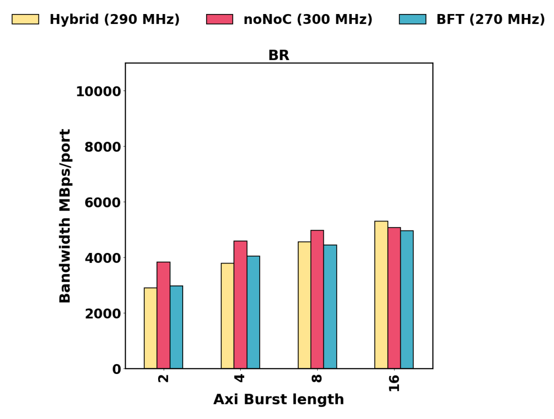
For single-flit designs (Fig. 5.6a), barring the Nearest-Neighbour addressing policy, the BFT-SF delivers the higher throughout out of all designs. The highest throughput speed of 2.2 was achieved for the Criss-Cross addressing policy. Similarly, for multi-flit designs (Fig. 5.7), both the NoCs BFT-MF and Hybrid-MF outperform the noNoC-MF design for all addressing policies, except Nearest-Neighbour and Bit-Reversal for burst lengths 2,4 and 8. For the Nearest-Neighbour addressing policy, the PEs access just the pseudo channel immediately next to it. It is comparable to P2P-1, and consequently, this experiment's results closely follow that of P2P-1, where the PE with the highest frequency delivers the highest throughput. Similarly, with the Bit-Reversal addressing policy, PEs with a palindrome base number behave the same as PEs with P2P-1 addressing policy, and only few PEs make cross-stack or cross-bank accesses. Therefore, the effectiveness of the multi-flit NoCs is poorer for Bit-reversal, and it delivers less throughput than the noNoC-MF design. For Criss-Cross and Tornado the PEs make more cross-bank and cross-stack accesses, and thereby the NoCs improve the achieved throughput.



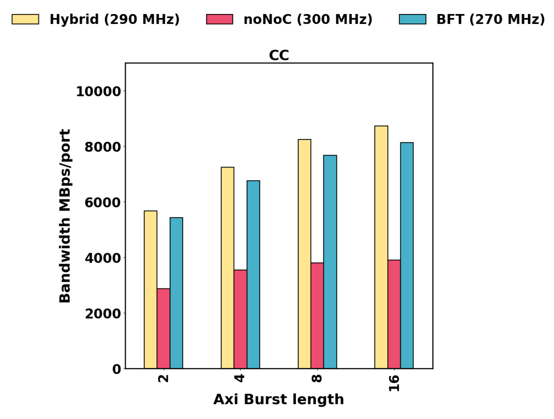
(a) Throughput vs Various Miscellaneous addressing policies for single flit designs

To summarize the results from our Synthetic benchmarking experiments, using Overlay NoCs improves the achievable throughput per port for the vast majority of the synthetic benchmarks. The throughput speedup is higher for single-flit designs, notably for CS-16 and P2P-24 addressing policies, a speedup of 9.8 and 8.4 is observed when the BFT-SF NoC is used. Consequently, for single-flit networks, BFT-SF emerges as the clear winner for synthetic benchmarks. In contrast, the Hoplite-SF suffers from the limited number of pending read requests that can be made, imposed for guaranteeing a livelock-free operation.

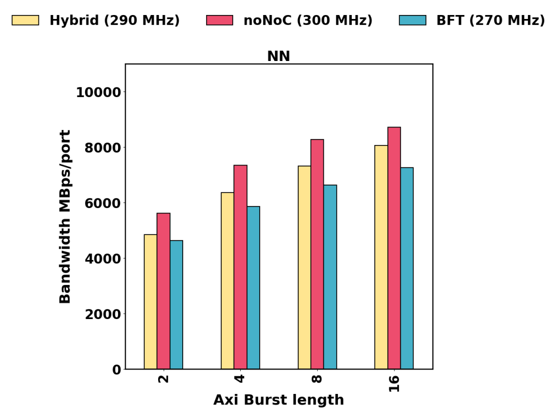
With multi-flit designs, as the burst length increases, the effectiveness of our NoCs decreases. For lower burst lengths such as 2, 4 and 8, our NoCs output performs the noNoC-MF design across all addressing policies. For instance, with CS-16 and P2P-24 addressing policies, the Hybrid-MF delivers a throughput speed up of 2.6, and 4.4 respectively, with the burst length set to 2. However, with burst length 16, the throughput gains are lower, and the throughput speedup drops to 1.7 and 1.26. Nonetheless, Hybrid-MF emerges as the winner for multi-flit networks for the vast majority of the synthetic benchmarks.



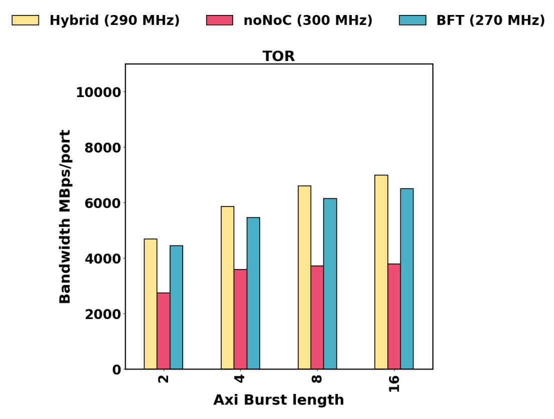
(a) Throughput vs Bit-Reversal Addressing policy



(b) Throughput vs Criss-Cross Addressing policy



(c) Throughput vs Nearest-Neighbour Addressing policy for multi-flit design



(d) Throughput vs Tornado Addressing policy

Figure 5.7: Throughput vs Various Miscellaneous addressing policies for Multi-Flit designs

## 5.4 Application Benchmarks

In this section, we present the results of our Dense Matrix Matrix Multiplication and Sparse Matrix-Vector Multiplication benchmarking.

| Matrix Size | NoC Type | Frequency (MHz) | Tput/port (MBps) | Tput Utili. (%) | Runtime (ms) |
|---|---|---|---|---|---|
| **17920** | **noNoC** | 254 | 4217 | 54.42 | 44462.4 |
| | **Hybrid** | 256 | 5393.29 | 69.03 | 84055 |
| | **BFT** | 251 | 5376.3 | 70.18 | 70431.9 |
| **8064** | **noNoC** | 254 | 4220 | 54.44 | 4043.38 |
| | **Hybrid** | 256 | 5417.47 | 69.34 | 7631.69 |
| | **BFT** | 251 | 5310.94 | 69.33 | 6372.62 |
| **2048** | **noNoC** | 254 | 4301.81 | 48.87 | 0.753235 |
| | **Hybrid** | 256 | 5322 | 66.5 | 0.718161 |
| | **BFT** | 251 | 5115.55 | 70 | 0.68355 |

Table 5.2: Results from DMM benchmarking experiment

### 5.4.1 Dense Matrix Matrix Multiplication

For DMM benchmarking experiment, we used 16 PEs with different input matrix sizes of dimensions 2048×2048, 8064×8064 and 17920×17920. For all the inputs, the results from the PEs were verified to be correct using the host code. The results of this experiment are tabulated in Table 5.2. It can be observed that while both the BFT and Hybrid NoC improve the throughput utilization per port by at most 1.29 times the throughput without the NoCs, the latency of the multiplication is significantly higher than the noNoC design. In the worst case, PEs with Hybrid NoC takes 1.9 times the time taken to complete a workload of matrix size 17920 than without any NoC. From this, we can infer that with DMM PEs, the throughput increase obtained by using NoCs is offset by the contention and additional cycles taken for the flits to travel through the NoC. Furthermore, the table shows that the runtime performance with the NoCs gets poorer with the increase in the matrix size. As the matrix size increases, the NoC gets congested with packets from all the PE, and as a result, the packets spend more time in the network, further worsening the latency.

From this experiment, we can conclude that for an application with a uniform memory access pattern such as DMM, using the Overlay NoCs to interface such PEs to the HBM is not ideal.

### 5.4.2 Sparse Matrix Vector Multiplication

We used 24 PEs to compute SpMV for *dw8192*, *epb*, and *t2d_q9* benchmark matrices for this experiment. The iteration count was set to 2048000, and the result was verified to be correct using the host code. The results of this experiment are tabulated in Table 5.3.

It can be observed that both the BFT and Hybrid NoC offer up to 1.40× and 1.43× speedup in runtime, respectively, when compared to the noNoC design. Both the NoC perform better as the sparsity increases. However, the BFT wins over the Hybrid for all benchmarks barring the t2d_q9 benchmark. The Hybrid NoC's runtime is marginally lower for the t2d_q9 benchmark. BFT's better performance than the Hybrid NoC can be attributed to the architecture of the Hybrid NoC. BFT offers the highest bi-section bandwidth of $O(N)$ for PE to PE communication. In the Hybrid NoC, we replace the top two levels of the Pi switches with several ring NoC, lowering the bi-section bandwidth for those two levels. Therefore, in the Hybrid NoC, any communication between PEs in different rows will have lower throughput than BFT NoC.

Similarly, Hybrid NoC offers the highest throughput per port, as reported by Vitis. But, if we consider the throughput utilization per port corresponding to the operating frequency, BFT NoC beats the Hybrid NoC, and both the NoCs roughly improve the throughput utilization per port by at least x1.92.

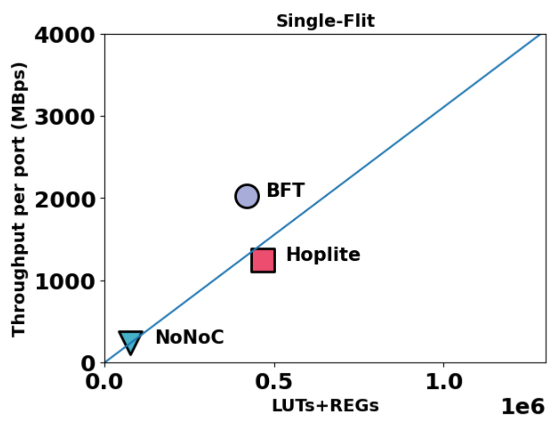
To summarize, the NoCs improve the throughput utilization per port for both DMM and SpMV. With SpMV PEs, using the NoCs improves the throughput utilization times, and a runtime speedup of 1.4 times is achieved. With DMM, NoCs improve the throughput by 1.29 times, by the latency is worse than noNoC design. This increase in latency with the NoC is attributed to the increase in the contention within the network and the regular memory access pattern of the DMM PE.

| Benchmark | No. NZV | NoC Type | Frequency (MHz) | Tput/port (MBps) | Tput Util.(%) | Runtime (ms) | Speedup | Data Transfer/port (MB) |
|---|---|---|---|---|---|---|---|---|
| dw8192 | 41746 | noNoC | 174 | 2624.23 | 49 | 68802 | - | 73,728 |
| | | Hybrid | 236 | 6403.15 | 88 | 51019 | 1.34 | 0.0508 |
| | | BFT | 203 | 5677.1 | 91 | 49030 | 1.40 | 0.0508 |
| epb | 95053 | noNoC | 174 | 2512.03 | 47 | 117189 | - | 122,880 |
| | | Hybrid | 236 | 6615.37 | 91 | 87412 | 1.34 | 0.097 |
| | | BFT | 203 | 5831.25 | 94 | 85877 | 1.36 | 0.097 |
| t2d_q9 | 87025 | noNoC | 174 | 2647.07 | 47 | 96241 | - | 98304 |
| | | Hybrid | 236 | 6494.53 | 91 | 67029 | 1.43 | 0.0729 |
| | | BFT | 203 | 5722.12 | 94 | 68309 | 1.40 | 0.0729 |

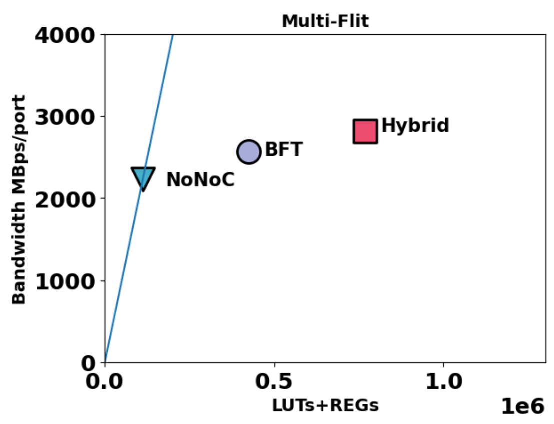
Table 5.3: Results from SpMV benchmarking experiment

## 5.5 Resource Utilization

In Table 5.4, we tabulate the breakdown of resources required by the NoCs (including the HBM-NoC interface module) and the resource consumed by the synthetic PE. For noNoC designs, we tabulate the resource consumed by the synthetic PE and HBM-NoC interface modules required to conduct the experiment. Since the Vitis instantiated modules are

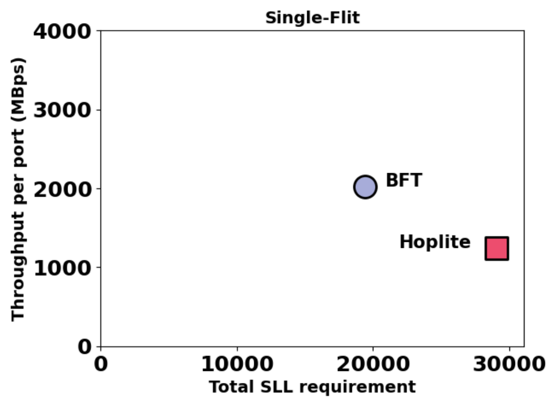


(a) Throughput vs LUTs+REGs for P2P-24, for single-flit network

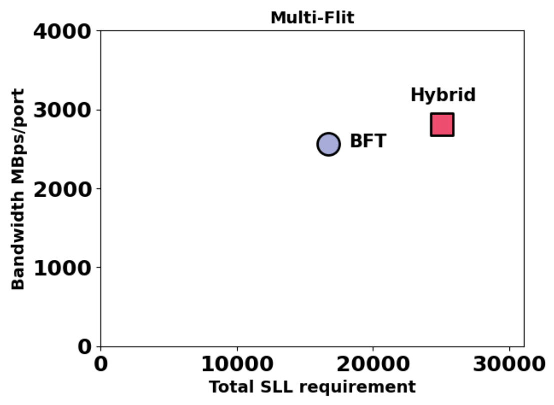


(b) Throughput vs LUTs+REGs for P2P-24, for multi-flit network, burst length = 16

Figure 5.8: Throughput vs LUTs+REGs



(a) Throughput with P2P-24 addressing policy vs Total SLL requirement , for single-flit network



(b) Throughput with P2P-24 addressing policy vs Total SLL requirement for multi-flit network, burst length = 16

Figure 5.9: Throughput vs SLL requirement

common for both with and without NoCs, there are ignored. In Figure 5.8, we plot the attained throughput per port for P2P-24 addressing policy vs resource utilization for our designs. From Figure 5.8a, it can be observed that the BFT-SF offers better Throughput per LUT+REG spent than noNoC-SF and Hoplite-SF.

However, this win is lost for multi-flit designs, as shown in Figure 5.8b. As discussed in Section 5.3, the effectiveness of our networks drops with the increase in burst length, and the presence of Virtual channels in Hybrid NoC requires buffers with pipelined inputs are in every ring switch, resulting in higher resource utilization. It consumes 1.8 times the LUTs+REGs consumed by BFT-MF NoC without a commensurate increase in throughput. Consequently, both the BFT-MF and Hybrid-MF offer less Throughput per LUT+REG than noNoC-MF design for P2P-24 with burst length 16.

| NoC Type | LUT | REGs |
|---|---|---|
| **BFT-SF** | 164,034 (12.59%) | 255,923 (9.82%) |
| **Hoplite-SF** | 172,105 (13.2%) | 293,356 (11.25%) |
| **noNoC-SF** | 26,784 (2.05%) | 48,592 (1.86%) |

| NoC Type | LUT | REGs |
|---|---|---|
| **BFT-MF** | 169,143 (12.9%) | 254,380 (9.76%) |
| **Hybrid-MF** | 288,583 (22.15%) | 478,732 (18.37%) |
| **noNoC-MF** | 43,704(3.35%) | 68,864(2.64%) |

Table 5.4: Resource Utilization Breakdown for Multi-Flit and Single-Flit design

In Figure 5.9, we plot the throughput achieved with P2P-24 addressing policy vs the total SLL requirement for multi-flit and single-flit designs. From Figure 5.9a, it can be observed that BFT-SF offers higher throughput and consumes less SLL (SLR0-SLR1 and SLR1-SLR2) than the Hoplite-SF. Similarly, from Figure 5.9b it can be observed that the Hybrid-MF outperforms BFT-MF but also consumes more SLLs. In both these designs, the number of PEs assigned to the BFT-SF (by extension BFT-MF) is different from the Hoplite-SF and Hybrid-MF. Therefore, while a direct comparison might not be possible, the plot helps gauge the performance of the noNoC with respect to the total SLLs consumed by them.

However, this image does capture the subjective ease of using Hybrid-MF or Hoplite-SF for floorplanning. The addition of PEs or moving PEs from one SLR to another is relatively more straightforward in Hybrid and Hoplite than BFT; the SLL utilization remains the same. Whereas, with BFT, several design choices must be examined to find the right cut that balances the congestion and SLL consumption. This also entails several directed changes in the RTL to instantiate SRL FIFOs in the desired switches at the desired levels.

# Chapter 6

# Conclusions

FPGAs have become a popular platform for hardware acceleration alongside GPUs. However, traditional memory solutions in FPGA such as BRAM and DDR RAM either lack the storage capacity or the throughput capacity. HBM aims to satisfy the growing need for a memory that offers high throughput and storage capacity. It offers high throughput of up to 460 GBps and a memory capacity of 8GB. Nevertheless, due to the deficient internal crossbar, the throughput delivered drops by 3 to 10 times for full global access. Furthermore, in Alveo U280, all of the HBM ports are available only in one die, forcing the designers to utilize difficult to use SLLs to distribute the HBM bandwidth to other dies. In this thesis, we proposed using Overlay NoCs such as BFT and Hoplite to optimize the HBM throughput by supplanting the need for the deficient internal crossbar and use the chip-spanning capabilities of the NoC to distribute the HBM bandwidth to other dies. Observing the shortcomings of BFT and Hoplite NoC, we combined select features to propose a new Hybrid NoC to meet our objective.

To test the efficacy of NoCs in optimizing the HBM bandwidth, we designed synthetic and application benchmarks such as Dense Matrix Multiplier and Sparse Matrix-Vector Multiplier. We conducted several benchmarking experiments with and without NoCs. Our experiments show that using Overlay NoCs improves the HBM throughput per port. Notably, with synthetic benchmarks, for single-flit global access (P2P-24), BFT NoC improves the throughput by 8.6 times and 9.8 times for cross-stack access with radius of 16. However, the throughput gains offered by our networks drop with the increase in the burst length. Nevertheless, Hybrid NoC with burst length 2 improves the throughput by 2.6 and 4.4 times for P2P-24 and CS-16, respectively. With burst length 16, Hybrid NoC improves the throughput by 1.7 and 1.26 times for the same access patterns.

Our NoCs also improve the frequency of resource-intensive PEs such as DMM and SPMV by enabling chip-spanning designs. Notably, both BFT and Hybrid NoC improve the operating frequency of SPMV PE from 174 MHz to 203 MHZ and 236 MHZ, respectively.

## 6.1 Future Work

This section lists the possible avenues for future work based on the work done in this thesis.

1. **Sorting Benchmark**. Similar to the DMM and SpMV, Sorting hardware acceleration in FPGA is a popular application in FPGA. Therefore, testing the efficacy of our Overlay NoCs with sorting benchmarks such as Merge and Bucket Sorting PEs is essential. Similarly, other popular hardware acceleration applications such as FFT and Image processing IPs could also be considered.

2. **AXI to AXIS interface** AXI protocol is the popular interface for many FPGA-related applications and accelerators. Therefore, it is essential to support PEs with Full-AXI interfaces. A potential solution is to design and implement an AXI to AXIS interface that converts the messages sent through the AXI protocol into a form factor suitable for AXIS protocol (and vice-versa) used in our NoC.

3. **Improving Floorplanning**. BFT NoC suffers from significant congestion due to the nested floorplanning. Improving the floorplanning and implementing the necessary design changes would significantly impact the operating frequencies of BFT NoCs.

4. **Scalability**. 24 PEs were used in SpMV and Synthetic benchmarks in our experiments, whereas 16 PEs were used in the DMM benchmark. A possible research avenue is to increase the number of PEs in the workload and study the impact on operating frequency and the throughput per port.

5. **Intel Stratix 10**. All of our experiments were done with Xilinx Alveo U280. HBM ports in Alveo U280 are exposed to only one die, whereas the HBM in Intel Stratix 10, the HBM ports split across different dies, where each die gets some HBM ports. This change in how HBM ports are exposed to the fabric brings new challenges and possibilities. A possible research avenue would be to adopt the proposed NoC designs for Intel Stratix and examine its feasibility.

# References

[1] Shereen Afifi, Hamid GholamHosseini, and Roopak Sinha. Hardware acceleration of svm-based classifier for melanoma images. In Fay Huang and Akihiro Sugimoto, editors, *Image and Video Technology – PSIVT 2015 Workshops*, pages 235–245, Cham, 2016. Springer International Publishing.

[2] Amin. *System-on-a-chip (soc)-based hardware acceleration for human action recognition with core components*. PhD thesis, 2018.

[3] Rajeev Balasubramonian. Lecture 13: Interconnection networks - university of utah.

[4] A. Becher, D. Ziener, K. Meyer-Wegener, and J. Teich. A co-design approach for accelerated sql query processing via fpga-based data filtering. In *2015 International Conference on Field Programmable Technology (FPT)*, pages 192–195, 2015.

[5] Maciej Besta, Dimitri Stanojevic, Johannes de Fine Licht, Tal Ben-Nun, and Torsten Hoefler. Graph processing on fpgas: Taxonomy, survey, challenges. *CoRR*, abs/1903.06697, 2019.

[6] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. Hbm connect: High-performance hls interconnect for fpga hbm. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 116–126, New York, NY, USA, 2021. Association for Computing Machinery.

[7] Nicola Concer. *Design and performance evaluation of Network-on-chip communication protocols and architectures*. PhD thesis, Universit'a di Bologna, 2009.

[8] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[9] Dally and Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[10] James W. Dally and Brian Towles. *Principles and practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2003.

[11] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[12] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, 2001.

[13] Johannes de Fine Licht, Maciej Besta, Simon Meierhans, and Torsten Hoefler. Transformations of high-level synthesis codes for high-performance computing. *CoRR*, abs/1805.08288, 2018.

[14] M. N. Emas, A. Baylis, and G. Stitt. High-frequency absorption-fifo pipelining for stratix 10 hyperflex. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 97–100, 2019.

[15] Shane T. Fleming, Ivan Beretta, David B. Thomas, George A. Constantinides, and Dan R. Ghica. Pushpush: Seamless integration of hardware and software objects via function calls over axi. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2015.

[16] Tzanis Fotakis. *Analysis and design methodology of convolutional neural networks mapping on reconfigurable logic*. PhD thesis, Technical University of Crete, 2020.

[17] Emmanouil Fragkakis. *Deadlock avoidance with virtual channels*. PhD thesis, 2009.

[18] Tushar Garg. *Hoplitebuf FPGA network-on-chip: architecture and analysis*. PhD thesis, University of Waterloo, 2019.

[19] Roman Gindin, Israel Cidon, and Idit Keidar. Noc-based fpga: Architecture and routing. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 253–264, 2007.

[20] Jan Gray. *2GRVI Phalanx: A 1332-core RISC-V RV64I Processor Cluster Array with an HBM2 High Bandwidth Memory System, and an OpenCL-like Programming Model, In a Xilinx VU37P FPGA [WIP Report]*, 2018.

[21] Cristian Grecu, Andre Ivanov, Partha Pande, Axel Jantsch, Erno Salminen, Umit Ogras, and Radu Marculescu. Towards open network-on-chip benchmarks. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 205–205, 2007.

[22] R. Holsmark and S. Kumar. Design issues and performance evaluation of mesh noc with regions. In *2005 NORCHIP*, pages 40–43, 2005.

[23] Abhishek Jain, Hossein Omidian, Henri Fraisse, Mansimran Benipal, Lisa Liu, and Dinesh Gaitonde. A domain-specific architecture for accelerating sparse matrix vector multiplication on fpgas. 09 2020.

[24] Nachiket Kapre. Deflection-routed butterfly fat trees on fpgas. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2017.

[25] Nachiket Kapre and Jan Gray. Hoplite: Building austere overlay nocs for fpgas. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2015.

[26] Nachiket Kapre and Jan Gray. Hoplite: A deflection-routed directional torus noc for fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 10(2), March 2017.

[27] Nachiket Ganesh Kapre. *Packet-switched on-chip FPGA overlay networks*. PhD thesis, California Institute of Technology, 2006.

[28] Kaan Kara, Christoph Hagleitner, Dionysios Diamantopoulos, Dimitris Syrivelis, and Gustavo Alonso. High bandwidth memory on fpgas: A data analytics perspective, 2020.

[29] H.T. Kung and A. Chapman. The fcvc (flow-controlled virtual channels) proposal for atm networks: a summary. In *1993 International Conference on Network Protocols*, pages 116–127, 1993.

[30] Junghee Lee, Chrysostomos Nicopoulos, Sung Joo Park, Madhavan Swaminathan, and Jongman Kim. Do we need wide flits in networks-on-chip? In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 2–7, 2013.

[31] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[32] G. S. Malik and N. Kapre. Enhancing butterfly fat tree nocs for fpgas with lightweight flow control. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 154–162, 2019.

[33] George Michelogiannakis. *Approaching Ideal NoC Latency with Pre-Configured Routes*. PhD thesis, University of Crete, 2007.

[34] Philippos Papaphilippou, Jiuxi Meng, and Wayne Luk. High-performance fpga network switch architecture. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 76–85, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Charles Seitz and Wen king Su. A family of routing and communication chips based on the mosaic. In *In Proceedings of the University of Washington Symposium on Integrated Systems*, pages 320–337. MIT Press, 1993.

[36] A. Y. Seydim. Wormhole routing in parallel computers. 2001.

[37] A. Shawahna, S. M. Sait, and A. El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2019.

[38] Kevin Tran, Paul Silvestri, Bill Isaacson, Brian Daellenbach, and Chris Browy. *Start Your HBM/2.5D Design Today*, 2016.

[39] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. Benchmarking high bandwidth memory on fpgas, 2020.

[40] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. Shuhai: Benchmarking high bandwidth memory on fpgas. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 111–119, 2020.

[41] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. Hoplitert: An efficient fpga noc for real-time applications. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 64–71, 2017.

[42] Xilinx. *WP45 High-Performance, Lower-Power Memory Interfaces with the UltraScale Architecture WP454*, 2015.

[43] Xilinx. *UG 1270 Vivado HLS Optimization Methodology Guide*, 2017.

[44] Xilinx. *UG1037 Vivado AXI Reference*, 2017.

[45] Xilinx. *Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance*, 2019.

[46] Xilinx. *Alveo U280 Data Center Accelerator Card Data Sheet*, 2020.

[47] Xilinx. *Alveo U280 Data Center Accelerator Card User Guide*, 2020.

[48] Xilinx. *AXI High Bandwidth Memory Controller v1.0 LogiCORE IP Product Guide*, 2020.

[49] Xilinx. *Getting Started with Alveo Data Center Accelerator Cards*, 2020.

[50] Xilinx. *UG1393 Vitis Unified Software Platform Documentation - Application Acceleration Development*, 2021.

[51] Xilinx. *UG574 UltraScale Architecture Configurable Logic Block*, 2021.

[52] Xilinx. *UG904 Vivado Design Suite User Guide - Implementation*, 2021.

[53] Yi Xu, Bo Zhao, Youtao Zhang, and Jun Yang. Simple virtual channel allocation for high throughput and high frequency on-chip routers. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–11, 2010.

[54] Yong Yang Zou, Ming Chen, and Kang Lin Wei. Design of custom axi4 ip based on axi4 protocol. *Applied Mechanics and Materials*, 687-691:2326–2330, 2014.