A Network on Chip For Concurrent Transmission Of Accurate and Approximate Data

Richard Fenster

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

August 2022

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Richard Fenster**

Entitled: **A Network on Chip For Concurrent Transmission Of Accurate and Approximate Data**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair/Internal Examiner
Dr. Sofiène Tahar

_____ External Examiner
Dr. Rajagopalan Jayakumar

_____ Supervisor
Dr. Sébastien Le Beux

Approved by _____
Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

_____ 2022

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

A Network on Chip For Concurrent Transmission Of Accurate and Approximate Data

Richard Fenster

As time passes by, there is an increase in desired computational ability. Networks on chip have been seen to facilitate communication between tightly coupled processing elements located on the same die. With said increasing computational demand, comes an increase in power consumption. As a result, approximate computing techniques are used in conjunction with networks on chip to reduce power consumption and latency. When implementing these approximate networks on chip, additional resources are required and may not have maximal use during runtime. In this work, we propose a network on chip design that maximizes resource usage irrespective of data type. This is achieved by using virtual channels that are configured depending on one of two operating modes. An accurate only mode allocates all bandwidth such that a single accurate link is active. The other operating mode is a mixed mode where the bandwidth is halved for approximate and accurate data types. If operating in this mixed Mode, a latency reduction of up to 44.2% from baseline can be realized when approximate traffic accounts for sixty-seven percent of all network traffic. The additional power requirement for this improvement is 22.6% when wide flits are used.

Acknowledgments

First and foremost, I would like to thank Dr. Sébastien Le Beux, my supervisor, for willing to accept me as a graduate student. His insight and support has been invaluable in this journey, applying a methodological approach to steering me in the right direction when needed and pushing me to improve when needed.

I am also greatly thankful for the support from Mr. Ted Obuchowicz, VLSI/CAD Specialist, Dept. of Electrical and Computer Engineering, who always was there in times of need to act as a sounding board or lending an ear while making classic rock puns. As well, many hours of amusement have come about from discussing whether or not graphical interfaces should be abolished.

Additionally, I am indebted to Dr. Jelena Trajkovic, who was an assistant professor at Concordia University during my undergraduate studies and introduced me to the topic material of this thesis while providing guidance on my professional trajectory.

I am eternally grateful for the support of my parents and their patience throughout this journey. As well, Ziggy and Lacey, the most golden of golden retrievers who relentlessly beg for my attention.

Lastly, I would like to thank my friends who provided endless support through the ups and downs of this academic expedition. It's been a wild ride to say the least.

In memory of Dr. Terry Fancott. You, the reader, would not be reading this thesis if it weren't for his neverending kindness, optimism and willingness to give me a second-chance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the motivation of this thesis work is presented from both approximate computing and network on chip perspectives. Once established, the objective of the thesis is discussed, along with research contributions. Lastly, the structure of the thesis is presented.

1.1 Motivation

1.1.1 Approximate Computing

It has been well established that Moore's law, dictating that the number of transistors within an integrated circuit will roughly double every two years, has begun to slow down [1]. Researchers and product designers are seeking smaller transistor sizes in order to aggressively fit more into die sizes. Innovations such as FinFETs [2] have been found to potentially extend Moore's law to 3 nm. Research in the field of semiconductors currently points to sub three nanometer processes for device manufacturing [3] [4], however at the time of writing, commercial three nanometer processes have yet to come online at TSMC. When this trend is coupled with societal demands for more computational power on tap, especially in smaller form factors, this set of circumstances can pose to be problematic.

Researchers have sought to find ways to increase throughput, decrease power consumption and delay at both the hardware and software levels. Most software applications do have non-critical portions that can be subjected to approximation [5] and therefore can benefit.

Applications for approximate computing vary immensely. One set of prime candidate for approximation is image processing [6], [7] or video processing [8] where the human eye is resilient to noise and error, allowing for reduced storage requirements. Neural networks also benefit immensely from approximation as reducing arithmetic precision can lead to improvements in latency for neural networks [9], [10]. Other fields include but are not limited to scientific [11] and financial [12] computing.

Works have shown improvements in power consumption [13] by reducing computational accuracy at the software level but still retaining acceptable results, yielding up to a 30% decrease in power consumption. However, there is not one naive approximation method that benefits all use cases equally. This is just one of many reasons why the field of approximate computing grew in popularity. Approximations are performed in a slew of different mechanisms or techniques, each one providing some benefit in one or more aforementioned metrics while introducing error into the system. Consequentially, it is commonly argued that approximation techniques should be applied on a per application or profile basis [14].

1.1.2 Networks on Chip

To combat Moore's Law's stagnation, another means of improving power consumption and reducing latency is by tightly coupling different processing units together in a single die. This has led to the growth of the field of heterogenous computing and is ubiquitous in many products and designs today. Arguably, the first consumer grade exposure to heterogenous computing was through computer graphics and the announcement of Microsoft's DirectX 8.0 in November 2000 [15]. DirectX 8.0 provided a specification for programmable subunits on graphic processing units called pixel and vertex shaders to provide real time effects in video games.

As time went on, these subunits became referred to as shader units in a more generalized manner for Microsoft's DirectX 9.0C and attention to GPUs shifted solely from providing real time graphics to general computing tasks that year at SIGGRAPH [16]. nVidia had leveraged this specification in their products to produce C for Graphics, or known as Cg,

which became the foundation of what is known today is as general purpose graphics processing unit (GPGPU) programming through their CUDA toolkit. On the other hand, the Khronos Group which at the time was composed of many of nVidia's competitors were working on an open source direct competitor to CUDA known as OpenCL [17]. Both toolkits are used to harness the thousands of compute units (the evolution of shader units) for general computing tasks and kernels in fields such as finance [6] and machine learning [18]. To facilitate meaningful and efficient communication among these compute units, a network must be formed on the die.

The consumer marketplace for desktop graphics processors had networks on chips in some capacity since the early 2000s. However, the focus and use of NoCs were not just limited to data centers and desktop computing where gains were focused more so on performance than power consumption. nVidia's GeForce 6800GT, released in 2004, was the first GPU to have fully programmable shader cores and had a TDP of 67W on a 130nm process with 16 programmable cores [19]. Focus was turned towards the embedded market to capitalize on the throughput and power efficiency gains.

In 2008, nVidia had announced their Tegra System-on-Chip [20] which was a highly integrated module containing an ARM11 CPU, dedicated image and video processing cores, and a full fledged GeForce GPU capable of substantial graphics. nVidia had claimed to succeed in making a mobile system on chip that could support an HD camera, HD video decoding and other features. All these cores would have to be interconnected through some means, being a network on chip. At the time of announcement, most if, not all, mobile devices such as the Nintendo DS or the Apple iPhone had discrete components for processors, graphics, audio decoding and other peripherals. The SoC was not wildly successful but in the following year, nVidia had announced during the Mobile World Conference 2009 that they had achieved success in porting a budding smartphone operating system called Android to the Tegra. The following year, the Tegra 2 was announced with major smartphone developers such as Motorola, Samsung and HTC using it in their devices. It was at this point where a person could easily look no further than their pocket to have a device with a heterogenous system on chip with some form of internal interconnect linking different

processing elements.

In the consumer space, networks on chip are most commonly found in graphics processing units and associated variants such as GPGPU modules [21]. There is, however, an effort being made to integrate NoCs into consumer devices [22]. As well, ARM has provided AXI as a free to use interconnect that can be used to build networks on chip [23]. That said, in research-oriented bespoke implementations, networks on chip are more prevalent for design and scale from few cores up to having north of 500 full CPU cores on a single die [24].

1.2 Problem Statement

Approximate computing can lead to benefits in computational latency, throughput and reducing energy footprints of devices by introducing error-embracing techniques at either the hardware or software level. Some of these techniques can be extended or implemented for use at the network level for heterogeneous computing systems. If there is an overhead introduced for being able to implement approximate network on chip designs, this overhead is not considerably used. As discussed in [25], the de facto benchmark suite AxBench [26] will yield 20% of all total traffic to be approximate. Indeed, it can be assessed that approximate networks on chip suffer from efficient use of given resources at the cost of throughput. These pitfalls can be overcome by encouraging resource sharing in the design of the network on chip for both approximate and accurate data when transmitting both concurrently.

While these approximate networks typically offer gains in latency or energy efficiency, additional overhead is required to implement these devices at the network level. In a non-realistic scenario, all traffic throughout an approximate network would be approximate and there would be no need for protections or to be accurate. However, this is not the case as processing elements do require control and other types of packets that cannot be approximated. As a result, approximate networks on chip must support both the transmission of accurate/protected data and approximated data to be deemed useful.

A resource overhead is introduced because of this requirement of transmitting both

accurate and approximate data. However, approximate network on chip designs do not allow for the transmission of approximate and accurate data concurrently across the same link without requiring even more overhead to increase throughput. By having this additional overhead, any power consumption benefits may be reduced or worse, diminished.

1.3 Contributions

In this thesis, we propose a configurable approximate network on chip design that focuses on efficient resource use. The design may be operated in one of two modes. An "Accurate Mode" where the resources of a link are fully utilized in transferring accurate or protected data is one of two modes available. The other mode is a "Mixed Mode" where link resources and bandwidth are split in half to facilitate approximate and accurate transmission of packets, albeit with a latency penalty when processing accurate packets. We have found that with our design, a reduction up to 40% of global network traffic latency can be realized when operating in Mixed Mode and the majority of network traffic is approximate. To implement a network tile capable of Mixed Mode operation with a wide flit size (128 bits), a 26% resource overhead is introduced. Validation and implementation of the design is performed at the RTL level. A Xilinx Virtex Ultrascale+ FPGA is used for implementation and determining both resource use and power consumption. Mentor Graphics Modelsim is used to verify functionality as an RTL simulation with randomly generated traffic. Consequentially, our contributions have been presented at the 14th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2021) [27].

1.4 Thesis Structure

The thesis is structured as follows: first, a literature review is performed in Chapter 2. Several techniques within the broad paradigm of approximate computing are introduced such as precision scaling, bit field reduction and voltage scaling. As well, essential developments in the field of networks on chip are introduced, pertaining to topologies and routing

strategies. Once these fields have been introduced, a related works review of approximate networks on chip is presented, using approximate computing as an applied technique within networks on chip. The proposed design of RELAX is presented in Chapter 3, discussing each module within the network and how the network functions from a high level. Chapter 4 discusses the register-transfer level design of network components from the ground up, including basic components such as buffers. RELAX's network performance is discussed and evaluated in Chapter 5 along with characterizing an FPGA implementation in terms of resource usage and power consumption. Lastly, Chapter 6 concludes the thesis with applicable scenarios for use and potential future work.

Chapter 2

Related Works

2.1 Approximate Computing

2.1.1 Introduction

Approximate computing is a broad field of different techniques that range from the hardware realm to software. In this section, we will reduce our scope to solely hardware techniques. Mittal produced a survey work [5] that enumerates most of the techniques and applications of approximate computing, for both hardware and software, at the time of publication (circa 2016).

2.1.2 Inexact Hardware

The first type of approximate hardware to be considered is inexact hardware. This infers that the hardware intentionally produces inexact results to yield improvements in latency or power consumption. Most commonly, these research works involve manipulating either the floating point unit or ALU to obtain benefits.

In 1991, Wong et al. published a design for an approximate divider. At the core of the work, lookup tables and Taylor series approximations for the reciprocal are used to speed up division. An iterative algorithm is presented, using the reciprocal to find an approximate quotient. This approximate quotient is then multiplied by the divisor and the result is then subtracted from the dividend. This process is iterated until a result is obtained.

Pillai et. al introduced [28] a work in 1999, detailing the reduction in power delay product by scaling the precision of floating point computation in FIR filters. This is achieved by having a bypass mode to skip MAC units. However, these MAC units retain their state when entering the bypass mode, reducing dynamic power consumption. It was found that the resulting reduction in power delay product was north than 75%.

Kulkarni et al. introduced an inexact two bit by two bit multiplier in 2011 [29]. In this work, the multiplier returns `0b111` (7_{10}) as the result of `0b11` $\times$ `0b11` instead of `0b1000` (8_{10}). Kulkarni et al. found that by having a multiplier that produces 15/16 correct results and then using it to instantiate larger multipliers can lead to beneficial results. In particular, a JPEG encoder was designed using a resulting multiplier and was found to have savings in power consumption in the range of 31.78% to 45.4%, at the cost of 1.39% to 3.32% average error.

When designing adder circuits, the biggest hurdle in terms of delay is carry chain propagation. Kahng and Kang designed an inexact adder [30] in 2012 which is composed of partial adders without a carry chain. Should a carry be generated, the results are rendered inaccurate. The number of subadders can dictate the accuracy as needed. If greater accuracy is desired, the number of subadders can be decreased, resulting in an increase of how many bits each subadder handles. By doing so, the amount of carry chain gaps is reduced.

This type of work can be extended into advanced arithmetic blocks such as multiply and accumulate units [31]. Such implementations can be desirable in DSP units where energy savings can be paramount in the context of low power embedded machines. As Internet-of-Things devices become more popular, especially sensors and wearables, battery life can be extended as a result.

2.1.3 Precision Scaling

Approximation of floating point numbers can be achieved by manipulating the mantissa into being less accurate. This can be achieved through two different mechanisms: precision scaling and bit field reduction. Precision scaling aims to reduce the precision of a floating point operation by typically using smaller floating point units, leading to shrinking latency

by reducing computational complexity. As well, power consumption decreases due to having smaller FPUs. That said, a floating point value may still be represented in a (relatively) large format such as IEEE-754 [32] with a 32 bit mantissa that could have a sizable amount of its mantissa zeroed out. To augment the yielded improvements, bit field reduction can be applied, where the mantissa would effectively be either truncated or rounded down to a smaller number of bits. Using this method yields a smaller data structure than the standardized IEEE-754 32, 64 and 128 bit formats.

In 2007, Yeh et al. introduced their work [11] which details using a precision scaled physics simulation. In this work, profiling on a physics simulation is done before execution to determine the minimum required precision to sustain the simulation in a stable state. During runtime, the simulation is conducted with this predetermined precision level and a threshold checker to ensure that the simulation does not enter unstable bounds. Should the simulation reach this point, the precision is set to maximum in order to stabilize it. Once stable, the precision is then progressively lowered again. The goal was to to ensure the longest possible time with reduced precision then report on the energy savings.

Brunie et al. introduced a mixed-precision fused multiply and add unit for use in floating point units [33] in 2011. Fused multiply and add operations are defined as performing `Z = (A` $\times$ `B) + C`. The focal point of the work is using mixed precisions, specially that the multiplication operands are one level of precision and both the result and addend are doubly precise as the multiplication. As an example, `A` and `B` are 32 bit floating point numbers, while `Z` and `C` are 64 bit floating point numbers. Under normal circumstances, `A` and `B` would need to be represented in a 64 bit format to match the result and other operand's width. With some revisions to the datapath, it was found that the operation can be performed with mixed precision and no error. As a result, their mixed precision FMA unit is fit and compliant for use with Fortran and C based code using double precision.

Mach et al. introduced the `smallFloat` library of formats in their work, "A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing" [34]. In this work, the authors discuss the design of two custom floating point types. The first type is `binary8` which consists of a 3 bit wide mantissa and 5 bits allocated for the exponent. The

second is a 16 bit wide format called `binary16alt` that is complementary to the IEEE-754 2008 standard 16 bit wide floating point format. When using these formats, it was found that a performance gain of 15% to 25% could be achieved alongside an energy efficiency gain of 14% to 18%.

2.1.4 Approximation By Voltage Scaling

The last approximate computing technique of interest is voltage scaling. By reducing the voltage, power savings can be achieved typically at the cost of introducing bit errors. This technique is most commonly seen in implementations involving memory. One case can be found in DRAM implementations, where each cell storing one bit is composed of a capacitor holding the voltage level and a transistor acting as a gate. Due to the nature of DRAM cells, the stored charge will leak out over time, altering the stored value inside the cell. This, in effect, is a bit error and can be avoided by refreshing the cells in intervals. Jung et al. introduced the concept of approximate DRAM [35] by profiling the necessary refresh rates to retain a high degree of confidence in the data while allowing it to decompose. In this context, there is no latency or throughput gain. As a result, DRAM circuits consume less energy since the dynamic power is reduced by reducing the refresh interval.

Sampson et. al introduced analogous techniques for SRAM in [13]. The authors found that reducing supply voltages for SRAM devices introduces read upsets where bits may be flipped during a read operation along with write failures which are failures to write the exact value desired. By reducing SRAM supply voltages by 70%, which is deemed mild by the authors, the probability for a read upset is $10^{-16.7}$ and $10^{-5.59}$ for a write failure. A medium strategy for power reduction implies an 80% supply power reduction with read upset and write failure probabilities of $10^{-7.4}$ and $10^{-4.94}$. An aggressive strategy reduces supply power by 90% but introduces substantial increased risk for read upsets and write failures at 10^{-3} for both. With this in mind, the authors state a reduction in memory power of 17% for the mild strategy, 22% for the medium and lastly, 24% for the aggressive strategy.

Ataei and Stine further validate this concept of approximation on SRAM by focusing

their efforts in the context of video processing [8]. Their work introduces a 64Kb SRAM array that operates on three different supply voltages. It has been well established, as mentioned in their work, that for a given pixel in a video frame, human vision is most sensitive to the higher order bits of chroma and luma pixels. As a result, the most significant bits of chroma and luma pixels are stored using the highest of the three voltages, implying that these portions of the chroma and luma binary representations are protected from any bit errors that may be introduced by the SRAM. The middle chunk of the chroma and luma bit strings is stored by the middle supply voltage and is subjected to a moderate amount of bit errors and approximation. Lastly, the least significant segment of bits of each pixel is shown to have minimal perceptive difference in human vision and is therefore stored using the lowest supply voltage. Using this technique can yield up to a 69% improvement in power consumption.

2.1.5 Applications

The use of approximate computing in terms of applications or fields is immensely broad. One of the most immediately applicable fields for approximate computing is multimedia. This can seen in works such as [8] where approximation techniques are used to reduce the size of encoding chroma and luma pixels for video while maintaining satisfactory quality. As pixel densities for display devices increase, UHD resolutions such as 4K and 8K proliferate consumer devices. 4K Displays are more commonly found in televisions at cheaper price points and have begun to penetrate the mobile market in cellphones. 8K Displays, while available at a premium, are no longer a pipedream.

To provide rich content at these resolutions, the HEVC codec is commonly used [36]. Another example of approximation in multimedia is seen in [37] which details an approximate HEVC codec using fractional motion estimation, providing up to a 50% reduction in power consumption. Such an implementation would lend great value to mobile devices or other embedded systems such as hardware backends for television displays.

However, multimedia is not the only field that may benefit. Neural networks have seen use in the financial sector for more than two decades. The use of neural networks in this

sector range from fiscal early-risk warning [38], market prediction [39], GDP prediction [40] just to name a few. Depending on the complexity of these models, the latency of the network may be staggering. In these scenarios, inexact hardware or precision scaling can be used to improve the latency times of neurons.

AxBench [26] is a suite of benchmarks that is dedicated to quantifying approximate computing performance. As well, the PARSEC [41] benchmark suite has been ported for approximate computing benchmarking. Both AxBench and PARSEC cover similar areas in terms of applications. The included benchmarks in both span not just multimedia and machine learning, but numerical analysis, gaming among others.

2.2 Networks on Chip

2.2.1 Introduction

Networks on chip is an extremely diverse topic in computing research and as a result, the related works of interest pertaining to this thesis fall into two topics. Firstly, works involving network structure and topology are discussed. Afterwards, works focusing on network arbitration are presented. Within the scope of this section, it is assumed that the focus is on silicon networks on chip and not photonic or wireless variants.

2.2.2 Mesh Network Structure and Topology

When considering network topologies, the mesh topology is considered to be the most common due to its homogenous and simple structure. While other topologies such as ring and torus do exist, they are not considered within the scope of this thesis. One of the most fundamental and influential works is "Network on a Chip: An Architecture for Billion Transistor Era"[42], written by Ahmed Hemani et al. In this work, the authors lay down the foundation of networks on chip. Initially, a honeycomb structure is discussed, where neighboring nodes are interconnected along with a switch in the center of each honeycomb cell. This design requires two different routers, one for neighboring nodes and another for the honeycomb cell, handling six nodes. In 2002, Kumar et al. [43] presented their work,

"A Network on Chip Architecture and Design Methodology". This paper extends Ahmed Hemani's work by introducing the methodology for designing a mesh network on chip. A mesh router contains five ports, one for each cardinal direction (north, south, west and east) and lastly, one to connect to the processing element. In comparison to the honeycomb topology proposed, a mesh topology is simplified and far more regular, using one single switch as opposed to two independent switches.

Balfour et al. introduced the concept of a concentrated mesh topology in [44]. This topology is a modified mesh topology where each router connects to four processing elements and four routers. While this increases the latency of a router by one clock cycle, dynamic power consumption is less globally due to overall less hops needed. As well, since each router provides an increase in the number of processing elements connected to, the global latency of the network is also substantially decreased.

Yang and Wu presented their Tmesh topology [45] in 2010, serving as an improvement on the well-studied mesh topology. The mesh topology is modified by introducing a sixth and seventh port for the nodes on the edges of the network. These ports connect adjacent edges of the network in order to reduce the number of hops. Assuming a 3X3 mesh network, this implies that the worst case trajectory will always be at most two hops.

Due to its general simplicity, the mesh network topology is used as the basis for hybrid topologies. The first example of which is a mesh-torus implementation by Salah et al [46]. This topology is a classical mesh topology with links that wrap around at the extremities of a row or column. To implement this topology, there is little to no logic cost. However, there is increased effort when dealing with placement and routing for implementation to account for the signature torus link wraparound. In terms of global latency, this topology provides benefits for certain cases where a packet needs to traverse the entire width or length of the mesh.

Another hybrid topology is a ring-mesh design by Bourduas and Zilic [47]. In this work, a mesh network is divided into two sets of segments. The first and most atomic of these segments is a sub-mesh which consists of four nodes or tiles. In one of these tiles per sub-mesh, an interconnect can be found to interface with a ring of other local sub-mesh

networks. The collection of this ring and sub-mesh networks is referred to as a local mesh. As well, in the ring of a local mesh is a bridge that serves as an interconnect to a ring at the top of the hierarchy of multiple local mesh networks. As a result, this topology has strong priority for nodes to communicate within the sub-mesh and then the latency penalty increases substantially as packets travel beyond the sub-mesh. Assuming one node in a sub-mesh requires to send a packet to another sub-mesh contained within the same local mesh network, the packet must first travel to the node within the sub-mesh network that has the interconnect to the local mesh ring. Once it is received, it must then traverse the sub-mesh to reach its desired destination.

2.2.3 Resource Arbitration

When producing a network of any kind, the arbitration technique used is critical in determining the amount of latency. Typically, one of three strategies are used for resource arbitration. Before proceeding, two terms must be defined. Firstly, priority implies that a requester that desires to transmit a network packet is able to do so when resources are available irrespective of any other requesters. The second term is fairness, which is antagonistic to priority. Fairness implies that all requesters will be given an opportunity to transmit.

The first strategy of interest is round robin arbitration. With this strategy, all requesters are given equal priority and selection is determined by giving each requester the right to request the resource (in this case, the ability to transmit data outbound) in a fixed rotation. Xiaopeng et al. discussed this fundamental arbitration and its implementation in their work [48]. The issue with round robin arbitration is that there is no means of implementing priority and is a moderately fair arbitration technique. This implies that computational tasks may be stuck waiting, increasing latency while other packets are being transmitted.

The two other arbitration strategies implement some form of priority. A fixed priority encoding strategy is one where each requester is given a priority level. In the most simplistic of cases, each requester has a different priority level. The requester with the highest priority is automatically given the grant to transmit data when the resources to do so are available.

This type of arbitration is the cheapest in terms of digital logic to implement but also exhibits zero fairness.

The last arbitration technique is the matrix arbiter. It is seen as a compromise between fixed priority and round robin arbitration strategies. Matrix arbitration operates on a least-recently serviced strategy, allowing for strong fairness. As per Zhizhou and Xang [49], the matrix arbiter is the costliest to implement but scales far better with larger amounts of potential requesters. This implies that the least recently granted requester is given priority to transmit.

2.2.4 Flow Control and Routing Strategies

Flow control implies the mechanisms and signaling used to control the movement of packets from one node to another. Depending on the traffic of the network, the flow control mechanism and routing strategy used, an undesired state of the network called deadlock may be reached. Deadlock is defined as a state of the network where no data flow is possible and packet movement comes to an unrecoverable halt. A well implemented flow control algorithm will aid in preventing this state from being reached.

Though introduced for large scale computer networking, wormhole flow control is a commonly found mechanism used in the network on chip space. In wormhole flow control, a packet is split into segments called flits. During transmission of the packet, the receiving buffers are allocated for these flits. A characteristic property of wormhole routing is having multiple buffers in parallel with different packets. Each of these packets could, in theory, could be transmitted out of the order in which they were received. This functionality is known as virtual channels. Prakash and Ravikumar demonstrated the first VLSI implementation of a wormhole router with this functionality in 1994 [50]. The authors detail that by having the packet divided into flits, a header flit is typically produced and encoded to contain information such as the number of flits that follow and the destination.

As interest and research in networks on chip progressed, wormhole flow control was found to be an ideal candidate with the flexibility that can be obtained with virtual channels. However, one pitfall of wormhole flow control is if the buffers are entirely provisioned,

blocking any new data from being transmitted. This phenomenon is known as back-pressure and could ultimately lead to deadlock.

Due to the large amount of traffic that networks on chip may produce, this deadlock scenario became a major focal point in research. If a network is prone to deadlock, it is not a viable design. Because of this, major research efforts were exerted to overcome or prevent this hurdle. Palesi et al. published a work in 2009, detailing algorithms to be used when profiling applications [51] to be executed on a network on chip for producing ideal routing algorithms. While yielding deadlock-free results, this work will produce routing algorithms that are dependent on the applications being ran. In the following year, Seiculescu et al. published their work [52], detailing a method to remove deadlocks in networks on chip using wormhole flow control. In the work, the authors detail a universal means of removing deadlocks from any network topology by analyzing the network topology from a graph theory standpoint and determining which links are to succumb to deadlock. Once these points of failure have been determined, they are augmented with a minimal amount of physical or virtual channels. As a result of this processing, the authors managed to yield an approximately 88% reduction of resources needed for the network on chip. As a result of this resource reduction, a typical area savings of 66% was obtained.

Beyond the works of Seiculescu et al., there has been a large effort to find suitable means to validate mesh networks of various types to ensure that subsequent designs are deadlock free. In particular, Boppana and Chalasani presented a seminal work contained routing algorithms for fault-tolerant wormhole routing within mesh networks [53] in 1995, before networks on chip were considered viable. A year later, the authors had released an updated work that contained a framework to design said algorithms [54].

All things considered, not all network routing algorithms are bespoke and generated from algorithms based on expected network traffic. XY or YX routing is a common technique used to route packets throughout a mesh network. As implied in the name, XY routing will route a packet horizontally then vertically towards its' destination. YX routing behaves in a similar manner, however enforces vertical traversal for a packet first. Though these algorithms are particularly simple, they can be prone to deadlock as detailed by Zhang et

al. [55]. Zhang and company propose instead a routing algorithm called odd-even routing that is based on the odd-even turn routing model by Ge-Ming Chiu [56]. Zhang et al. argue that while their algorithm is more complex to implement, it is both adaptive and can be deadlock-free. Based on the results presented, odd-even routing provides an 11% increase in throughput for a 3X3 mesh network with a 4.6% average decrease in latency. That said, XY routing has been modified and augmented to prevent deadlocks as seen in [57], [58] and [59].

2.3 The Convergence: Approximate Networks on Chip

2.3.1 Overview

The design of networks on chip and their respective interconnects have converged with approximate computing in the past 15 or so years. This implies that approximation techniques are typically applied at the network physical layer, or OSI layer 1 [60]. Implementations and designs can be categorized in one of two categories. The first set is composed of lossy networks which infer some potential data corruption or loss of packets or flits. What results is that either flits or packets are dropped, or errors are introduced at the bit level that network interface will attempt to rectify the loss of data through some means of approximation. The second type is a lossless network, where flits or packets are not dropped but the contents are manipulated by the network interface to reduce flit size and the resultant packets.

2.3.2 Lossless Networks

Lossless networks are the more common of the two varieties and have been prevalent for longer. Lossless networks infer that, at a minimum, some form packet or flit size reduction is being performed by the network interface. In some cases, this attribute can be combined with additional approximation techniques are used before the network layer is considered.

Stevens et al. published their design, AxBA [61] in 2018. Though not a traditional network on chip but rather a bus architecture, the approximation techniques used are directly

applicable. AxBA aims to achieve reduce flit size through mathematical manipulation of data. Assuming a cache line of floating point values, AxBA transmits a baseline value followed by consecutive deltas of the values. This is known as base-delta compression. While the size of the flits is not reduced, this technique allows for more than one consecutive datum to be packed into a flit by only requiring the delta to be encoded.

Boyapati et al. introduced APPROX-NoC [62] in 2017 and is considered to be the paper to have established approximate networks on chip as a research interest. APPROX-NoC is a lossless approximate network on chip with a focus on pre-processing packets in order to compress them. The techniques for approximation are focused towards packet and flit size reduction. This is achieved by two mechanisms: if data can be approximated, as is the case with floating point numbers, the mantissa is manipulated through a logic block called VAXX so that the size is reduced. Afterwards, the data, irrelevant of being approximable or not, is fed to a compressor which then further reduces the flit size by mapping common bit patterns to smaller ones. Once the packet is received, it is reconstructed by the network interface.

DAPPER [63] was presented by Raparti et al. in 2018 with a focus on GPGPU designs. This design differs from most approximate network on chip designs since the approximation is not performed by the network interface itself or at OSI layer 1 at all. Instead, the approximation is performed by the memory controllers on the network.

Data approximation is performed if the core requesting the data from memory flags it as approximable. The network interface passes these flags as part of its request. If all values contained within a cache line can be approximated, the approximation process may begin. Each data packet sitting in the outbound buffer of the memory controller is analyzed to see if it can be merged into a single packet without exceeding the error threshold in order to reduce the number of packets transmitted.

Chen et al. presented an untitled approximate communication framework for networks on chips [64] in 2020. In this design, Chen et al. discuss a quality control framework to ensure that the approximations being performed stay within acceptable error ranges. A central quality control table is proposed that tracks memory addresses, the data type and

error tolerance associated. These values are obtained before execution of the application and require both profiling and preprocessing before runtime.

Based on the values of the table, data that can be approximated is subjected to first truncation based on the acceptable error threshold determined by preprocessing. Afterwards, further reduction in flit size is obtained by using pattern matching and encoding techniques similar to those found in APPROX-NoC [62].

2.3.3 Lossy Networks

In 2017, Wang et al. introduced ABDTR: Approximation-Based Dynamic Traffic Regulation for networks on chip [65] which is used as a stepping stone for other lossy network designs. This research work details a dynamic traffic regulation system that drops packets based a predefined quality control rate. A global controller issues each router an allowed drop rate to each router that is continuously updated based on network performance. With this value, a router may drop packets flagged as safe to approximate in order to reduce congestion. Upon reception of these safe to approximate packets, linear interpolation is used to reconstruct lost flits. Though more accurate and complex choices do exist for reconstructing data, linear interpolation offers a fair trade-off between implementation cost and quality of the end-product.

Ascia et al. discussed about the possibility of using a JPEG encoder as a tile on a network on chip with an approximate interface in [66], published in 2018. An apparent energy savings of 70% were realized with a root mean squared image degradation of 10^{-5}. This gain could be obtained if the payload flits going through the interface have a low supply voltage of 0.6V. Protected or control flits are transmitted with a high voltage of 1.1V.

Similarly, in 2018, Akram Ben Ahmed et al. presented AxNoC [67]. It is considered to be a seminal development in approximate networks on chip and introduced the concept of a lossy network. AxNoC functions by means of having the critical path of the network interface isolated such that it may operate with one of two supply voltages, a low and high supply voltage respectively. The higher supply voltage allows for protected transfers that are immune to bit errors or losses. When using the lower supply voltage, bit error rates

may be introduced into the transmission.

A caveat to having the dual supply system is that a certain amount of time is needed to set up and select the appropriate voltage source, which may introduce additional latency during transmissions. This can be seen especially when introducing protected flits or segments frequently. Though recovery methods are not used with AxNoc since whole flits or packets are not dropped, Ben Ahmed et al. have found that their design works well with a bit error rate of 10^{-5} to 10^{-7}.

Wang et al. introduced, as a follow up to ABDTR, AMNoC [68] in 2020, which is a multiplane network on chip consisting of a bufferless approximate layer with a light routing architecture and an accurate/protected layer. The approximate layer, or subnet, is bufferless and draws inspiration from SCARAB [69], a bufferless network on chip designed by Enright Jerger et al. When a packet is placed on the approximate subnet, it is subjected to bufferless transmission implying that flits may be outright dropped. Though AxNoC does not have any means of recovery since bit errors are the expected type of errors, AMNoC requires a recovery mechanism to account for flits to be lost. For this, Wang et al. chose linear interpolation once again as their means of recovery due to the lightweight cost to implement.

Xiao et al proposed ACDC [70] in 2020. This work adopts the mechanisms detailed in APPROX-NoC, in particular, VAXX. Therefore, ACDC uses the same mechanisms for approximation as APPROX-NoC. The focus of this work is not the technique to approximate data but however, a real time control of the quality of data versus network congestion.

A global controller analyzes the network congestion and is given a total error balance for the network. This error acceptance value is obtained by profiling the application being executed beforehand. It determines the error budget for serializing data and updates it accordingly during runtime based on heuristic analysis of network congestion and the error acceptance. The balance of the budget is then issued to each node, in which the node can independently determine how to spend that balance accordingly through approximating data. As well, if the network congestion is deemed to be reaching a critical point, flits may be dropped and then recreated by using nearest neighbor interpolation.

Chapter 3

System Level Description and Modes of Operation

3.1 Introduction

The approximate computing paradigm is one that encompasses many strategies and techniques to approximate data for some gain. As previously mentioned, these strategies can yield benefits in area, power consumption, latency and throughput [5]. In the context of networks on chip, power consumption of the network can be reduced through the use of dynamic voltage scaling techniques [67], reducing dynamic power consumption at the cost of potential increased packet drop rates or bit errors [13]. Both latency and throughput can be decreased through means of accuracy reduction. Accuracy-affecting approximation techniques such as using imprecise arithmetic [11] or loop perforation [71] do not have any effect on throughput or latency at the network level as these strategies are localized to processing elements. However, bit field reduction has a sizeable impact on latency and throughput at the network level [72] and can be implemented by modifying flit sizes.

While effective as per their functional mandates, the current variety of approximate networks on chip that use bit field reduction do not facilitate the concurrent transmission of approximate and accurate data. In this chapter, we introduce RELAX: a REconfigurabLe ApproXimate network on chip. This network on chip design can transmit both accurate and

bit field reduced approximate data concurrently with a focus on hardware reuse, minimal area and power overhead. RELAX itself does not perform any approximation but instead facilitates the transmission of reduced bit field approximate data.

RELAX is designed to operate in one of two modes. The first mode allocates the entire bandwidth of the network for accurate data transmission. If the network operates in the second mode, half the bandwidth is allocated to approximate, reduced bit field packets while accurate data transmission is given the remaining half. A net throughput increase can be obtained while operating in the second mode due to being able to transmit two packets concurrently.

First, the two operating modes provided by RELAX are introduced in detail and how these modes differ is discussed. Afterwards, the handshake protocol and link design used to transfer data from one network node to another is introduced in detail. Lastly, a functional description of the network at system level is presented.

3.2 Overview

3.2.1 Topology

For all intents and purposes going forward, the implementation and design of RELAX is focused on a two dimensional mesh topology. While RELAX can be adapted for use with other topologies such as torus and ring, the mesh was chosen as it is ubiquitous with networks on chip [73]. This implies that there are a total of five ports, one for each cardinal direction (north, south, west and east) and lastly, one for a processing element. Each link across two routers consists of two virtual channels referred to as (virtual) channels A and B, each N bits wide. As a result, a single direction link between two RELAX nodes may pass up to 2N bits of data in parallel. A three-by-three mesh RELAX network is shown in Figure 3.1, demonstrating packets being transferred in both modes. The mesh router used is presented in a generalized black box view in Figure 4.1, showing the links and configuration signals needed.

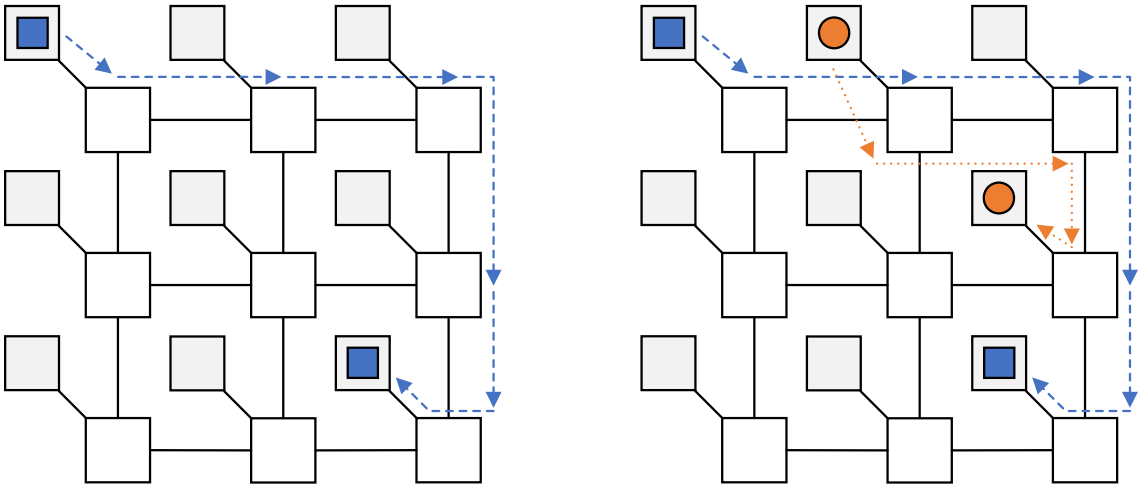


Figure 3.1: A 3X3 RELAX mesh network transmitting packets under different modes. When operating in Accurate Mode, an accurate packet utilizes 100% of the available bandwidth as shown on the left. Data transmission across the network is shown in Mixed Mode on the right. Accurate data is given half of the available bandwidth, while the remaining bandwidth is allocated to approximate data. This can be seen as an approximate packet, shown as a circle, is partially following the same trajectory as the accurate packet and being transmitted across routers at the same time. While concurrent transmission of both accurate and approximate data is possible, the accurate packet is penalized with an additional two clock cycle latency when operating in Mixed Mode in this example.

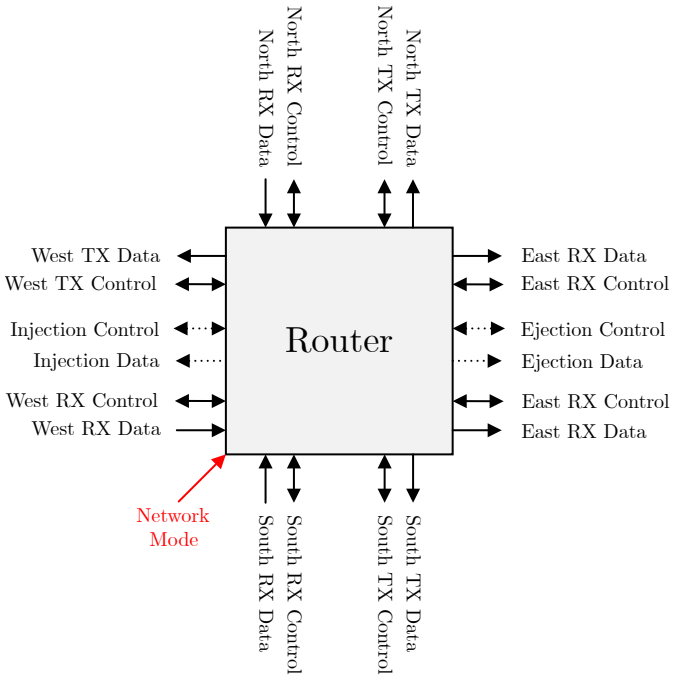


Figure 3.2: Black box representation of a RELAX mesh router.

3.2.2 Operating Modes

The desired operating mode is selected before an application's execution has begun. After a selection has been made, it is globally set throughout the entire network and may not change while the execution of an application takes place. An increase of data throughput when operating in Mixed Mode is achieved due to the design of RELAX. This is achieved by taking advantage of splitting network resources in half and the characteristic property of reduced size for approximate data [5].

When operating in Accurate Mode, RELAX uses both virtual channels in parallel as a singular channel for accurate data. As seen in Figure 3.3A, each datum AC_n occupies a slot in both virtual channels with higher order segments (AC_{nH}) being stored in Channel A and lower order segments (AC_{nL}) in Channel B respectively.

Mixed Mode grants the ability to transmit accurate and approximate concurrently using channels A and B respectively. Due to serialization, accurate data occupy two successive slots, AC_{nH} and AC_{nL}, in the buffer of Channel B. While this technique increases transmission latency for accurate data, Channel A is freely available to transmit approximate data independently of any accurate data packets. Each approximate packet is allocated one single slot per datum, as shown in Figure 3.3B. By doing so, approximate data transmission has an equivalent transmission latency of the network operating in the Accurate Mode.

3.3 Data Transfer and Protocol

At the core of RELAX is the network interface, which allows for processing elements such as memory controllers or processor cores to communicate across the network. Each interface has the ability to inject data into or eject data from the network. A network interface may inject data 2N bits wide when accurate and N bits wide for the approximate type.

It is assumed that the type of packet (accurate or approximate) to be transmitted is already determined by the processing element [74] and packaged accordingly. If the network is operating in the accurate only mode, both channels A and B have N bits of data injected

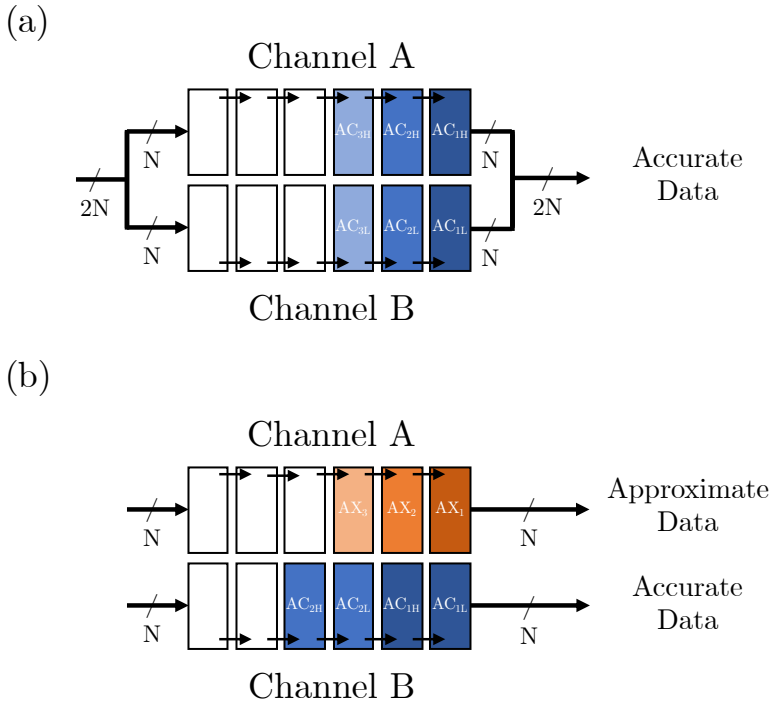


Figure 3.3: Virtual channel use in both operating modes.

into their FIFO buffers as seen in Figure 3.3A. The higher order bits are placed in Channel A, while the lower segment is stored in channel B. Should the network be operating in Mixed Mode, accurate data is placed in channel B since the buffer is equipped to write 2N bits of data in two adjacent N sized slots within one clock cycle. Figure 3.4A illustrates this functionality. The injection of approximate data is achieved by writing the higher order N bits to Channel A, as illustrated in Figure 3.4C.

The mechanism used by ejection ports is similar to injection ports. When operating in the Accurate Mode, both virtual channels are concurrently used to eject accurate data. During Mixed Mode operation, approximate data is ejected from Channel A, and accurate data from channel B. Since accurate data is stored across two FIFO slots during Mixed mode operation, the ejection port is equipped with the ability to deserialize the packet and eject the datum within a single clock cycle. It should also be noted that there is no priority to eject data based on type. This is done so that each network tile can schedule priorities as needed.

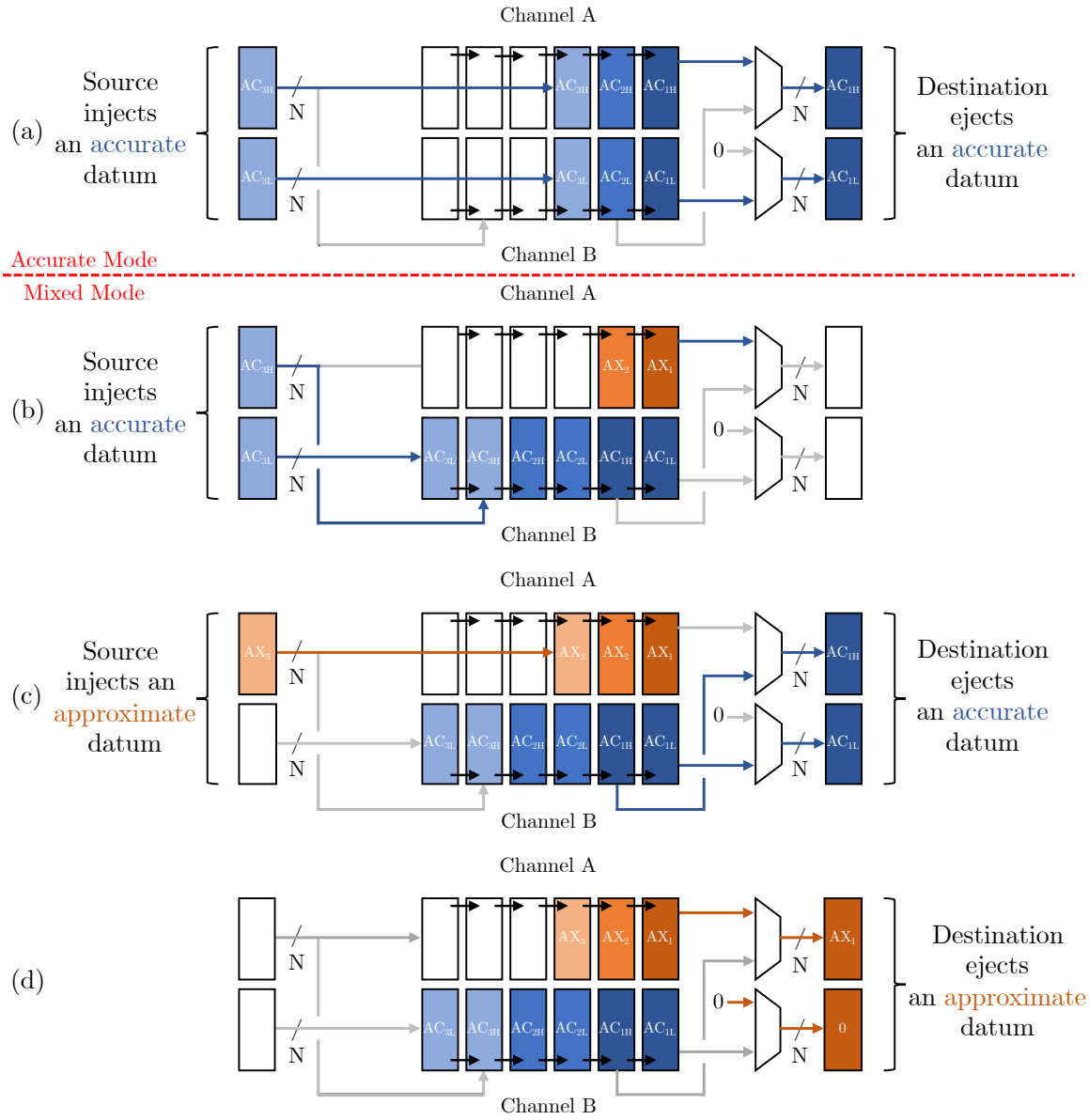


Figure 3.4: Visualization of data flow in both operating modes.

A lightweight protocol was chosen for the traffic flow since it must be implemented twice, as per the switching of operating modes. Since Mixed Mode splits the network into two effectively separate layers, any logic used for flow control must be minimal to reduce the amount of overhead. As a result of this directive, the RELAX protocol makes use of two flow control signals to implement its transmission protocol as a simple handshake [75].

The first signal, clearToSend, is sent by the receiver to the transmitter and indicates if

there is space within the receiving buffer to transmit data. If the buffer is full, this signal is set to a logic zero. The second signal, Valid, is set consequentially based on the state of both the receiver and transmitters' buffer occupancy. The connection and direction of these signals is shown in Figure 3.5a. Though there is one direction of data transfer shown for the sake of simplicity in the aforementioned figure, each link is in fact fully duplex.

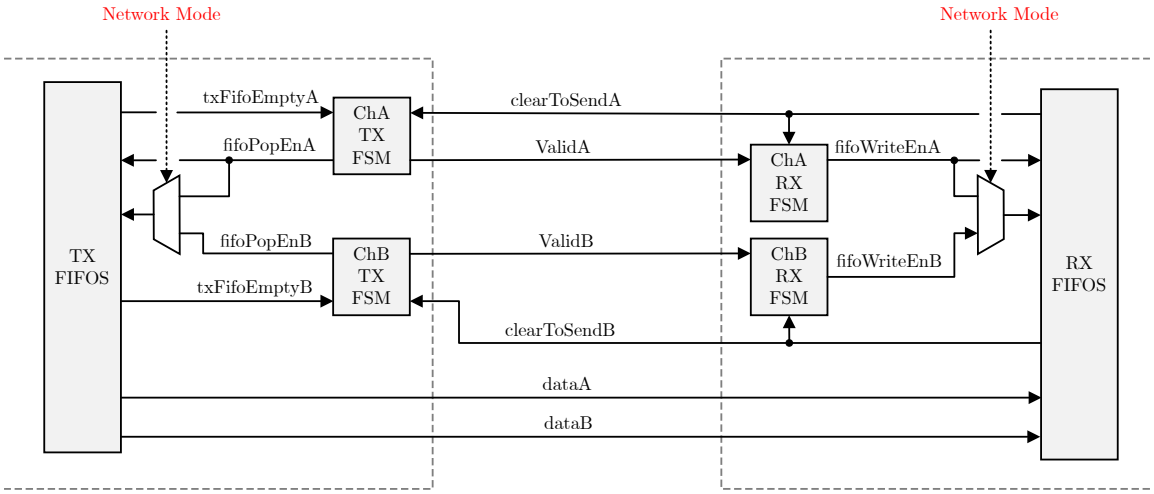
Should there be sufficient space at the receiver and there is valid data to transmit, Valid, is set to a logic one. During the following clock edge, the data is sampled by the receiver and popped from the transmitter's buffer. There are two pairs of these signals, one for each virtual channel. If the network mode is set to accurate only, the signal pair for Channel A is used to control the transmission. When running in Mixed Mode, each pair operates independently as each virtual channel operates separately. This behaviour can be seen in Figures 3.5b and 3.5c.

Traditionally, a three state handshake flow is used, consisting of ready, transfer and wait states [75]. As seen in Figures 3.5b and 3.5c, RELAX's transmission protocol only uses two states, being ready and valid. There is no need for a wait state since effectively, the TX finite state machines would simply set the Valid signal to zero just the same.

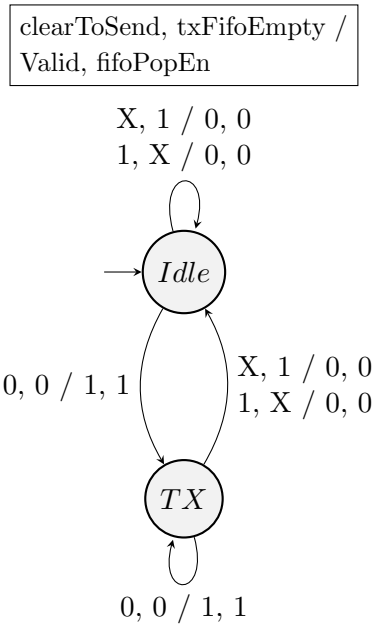
3.4 NoC System Level Design

3.4.1 Resource Allocation and Routing Finite State Machines

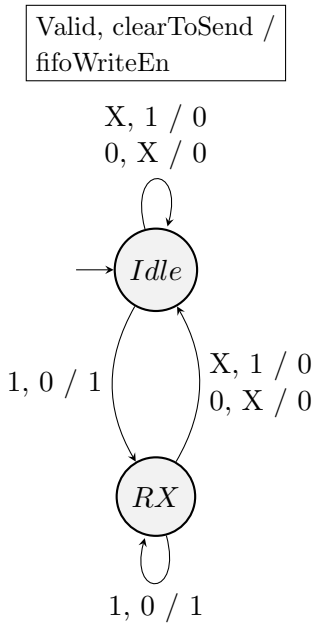
Traditionally, an arbiter is used to control resource allocation in NoC routers [49]. Since RELAX has a mode of operation that leads to splitting network resources into two independent networks, the mechanism used for arbitration must be carefully selected to meet the directive of encouraging hardware resource with minimal overhead. Because of this, a simple round robin FSM design is selected. As per the discussion in [76], a traditional arbiter that supports both modes would be classified as multi-fixed priority arbiters (MFPAs) and would been seen to have 10 inputs in the case of a mesh network (four cardinal directions and the processing element with accurate and approximate channels). As seen in Figures 10 and 11 of [76], having a 10 input MFPA can lead to a reduction of maximum frequency by



(a) Signals used for a single direction RELAX link.



(b) State diagram of TX FSM when operating in Accurate Mode and transmitting approximate data in Mixed Mode.



(c) State diagram of RX FSM when operating in Accurate Mode and receiving approximate data in Mixed Mode.

Figure 3.5: RELAX handshaking protocol signals and finite state machine flow. It is assumed for Figure 3.5c that if the networkMode input is set to a logic 1, the control of the channel B FIFOs is mirrored from those of channel A.

a factor of two and requires quadrupling the logic slices needed for a MFPA of five inputs, barring any additional overhead from handling concurrency and the different transmission requirements for both data types. With this realization in mind, it is quite apparent that

having an MFPA of 10 inputs does not meet the desired criteria of reducing the amount of overhead resources needed. The outcome of this revelation is a system of two minimal FSMs, where one FSM is active for both modes and another is engaged in parallel when operating in Mixed Mode.

The first FSM of interest is the Channel A, or accurate-mode FSM. It is designed to handle data transmission in the Accurate Mode and approximate data during Mixed Mode operation. It is delegated control of these operations due to the fact that both approximate and accurate only transmissions are one flit wide. Data transmission during the accurate only mode operation with this FSM is shown in Figure 3.6A. A multiplexer that feeds control signals to Channel B is present and is controlled by the network mode. When providing only accurate transmissions, it can be seen that Channel B is controlled by the accurate-mode FSM.

During Mixed Mode operation, the extra finite state machine (Mixed Mode FSM) is enabled via the multiplexer so that the routing of Channel B remains autonomous. Since accurate data is serialized and stored in two adjacent FIFO slots when using Mixed Mode, the Mixed Mode FSM allocates two clock cycles to each receiver for to pass the packet through the crossbar. Figure 3.6B visualizes the second clock cycle of an accurate data transmission, with the datum chunk AC_{1H} written to the appropriate transmitter. To account for having accurate data serialized and placed into two FIFO slots when running in Mixed Mode, the Mixed Mode FSM provides each receiver two clock cycles to pass data through the crossbar. The accurate data transmission shown in Figure 3.6B shows the second clock cycle as the datum chunk AC_{1H} is being passed to the transmitter.

Since the Accurate Mode FSM is designed to account for transfer of a packet in one clock cycle, it is repurposed when operating in the Mixed Mode. Through multiplexing, it instead controls approximate transmissions which require one clock cycle as opposed to the two cycles needed for accurate data transfers in this mode. This allows all resources to be used in the mixed mode as seen in Figure 3.4.

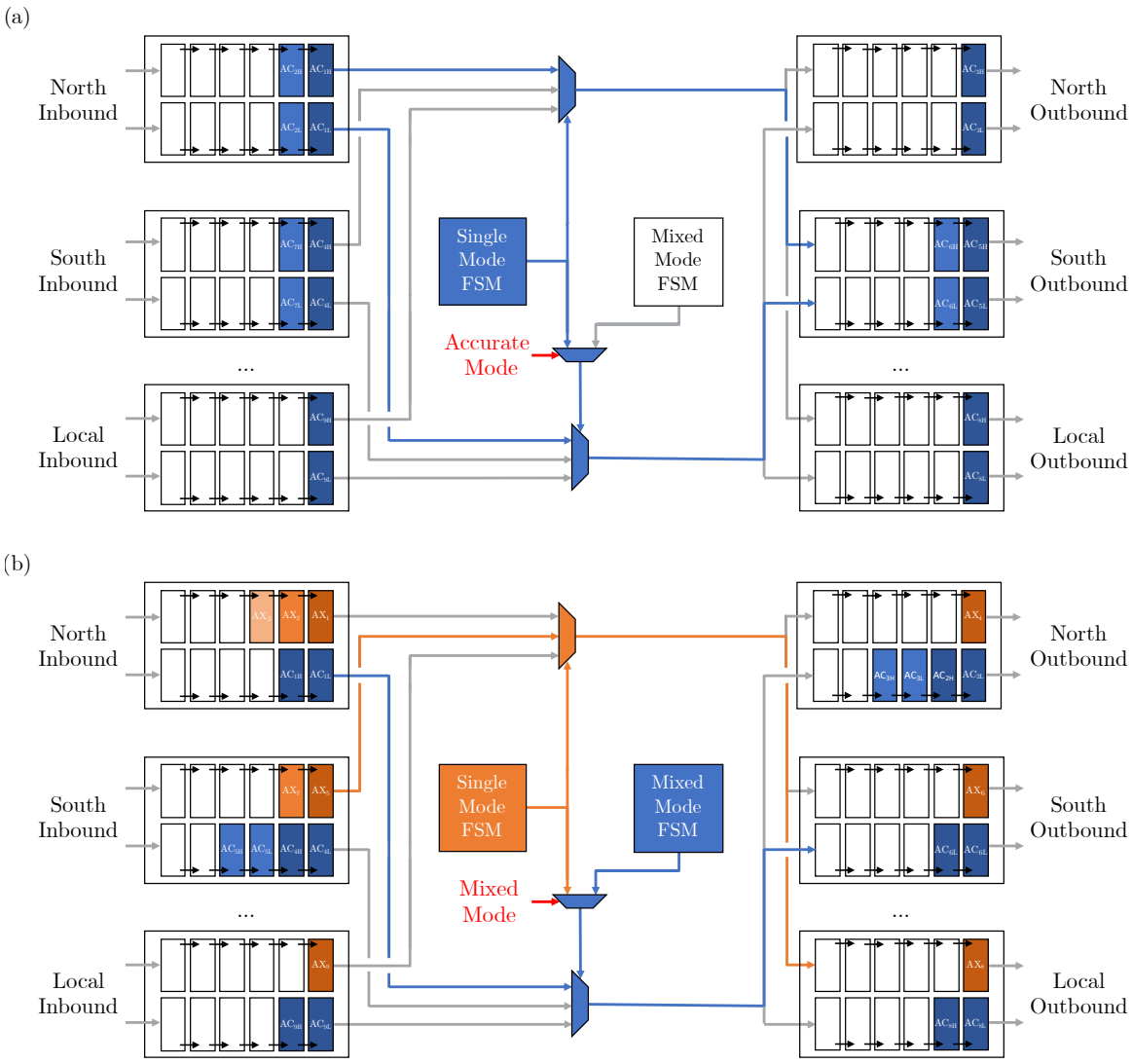


Figure 3.6: Differences in routing for both modes of operation.

3.4.2 Router Algorithm and Structure

Along with the network interface consisting of ejection and injection ports, a RELAX tile contains a router made up of a crossbar combined with receivers and transmitters for each link. The selected network mode dictates the behaviour of the crossbar and the router as a whole. When compared to regular NoCs, RELAX implements simultaneous and independent routing of both accurate and approximate data. Because of this, certain control resources within the router such as crossbars are duplicated. These extra resources

add to the design cost of RELAX. It can be seen in Figure 3.6 that the multiplexers of the crossbars are in fact controlled by the network mode selection.

In terms of routing algorithm, XY routing is selected. While Odd-Even routing provides better performance, XY routing requires substantially less resources to implement with a 14.2% latency penalty and a 10% penalty [55]. For the design goals of RELAX, this penalty is deemed acceptable since the penalty is invoked twice from implementing the mixed mode.

3.5 Summary

In closing, we present RELAX as a mesh network on chip that may operate in one of two network modes. These modes allow either transmission of either fully accurate data or a mixture of rounded/truncated approximate data and accurate data concurrently. Two virtual channels are used to provide this dual mode functionality with each data type delegated to one virtual channel when running in the mixed mode.

As well, the routing mechanisms and strategies used in RELAX are discussed. XY routing is the chosen routing strategy since it has a low implementation cost when compared to others such as odd-even routing. In terms of resource arbitration, the round robin technique is implemented through simple finite state machines in order to observe the tenet of reducing hardware overhead and resources. Another simple set of finite state machines are instantiated to implement the data transfer protocol which is a variation of flow control.

Chapter 4

RTL Design

4.1 Design Hierarchy and Overview

In this chapter, the RTL design of RELAX's components is presented along with the hierarchy of components. It is assumed that the components are implemented with a mesh topology in mind. With this topology, a tile is composed of five fully duplex links. Four links are used for the cardinal directions (north, south, west and east) to other network tiles. The remaining link is used to communicate with processing elements such as memory controllers or processor cores. At the network level, the tile is the top-most module instantiated and encapsulates numerous subcomponents. However, the tile itself does not encompass the processing element. The high level structure of a tile is shown in Figure 4.1.

We first discuss the router from a high level perspective. Afterwards, the different FIFO buffers used are discussed, which are essential to RELAX's functionality. Some of the buffers used in the network interface provide functionalities that are critical to the behaviour of RELAX depending on the selected network mode. The tile hierarchy is traversed, by focusing on link-level ports and their behaviours then routing logic blocks such as crossbars. Afterwards, the router is discussed at a module-level view along with the processing element port.

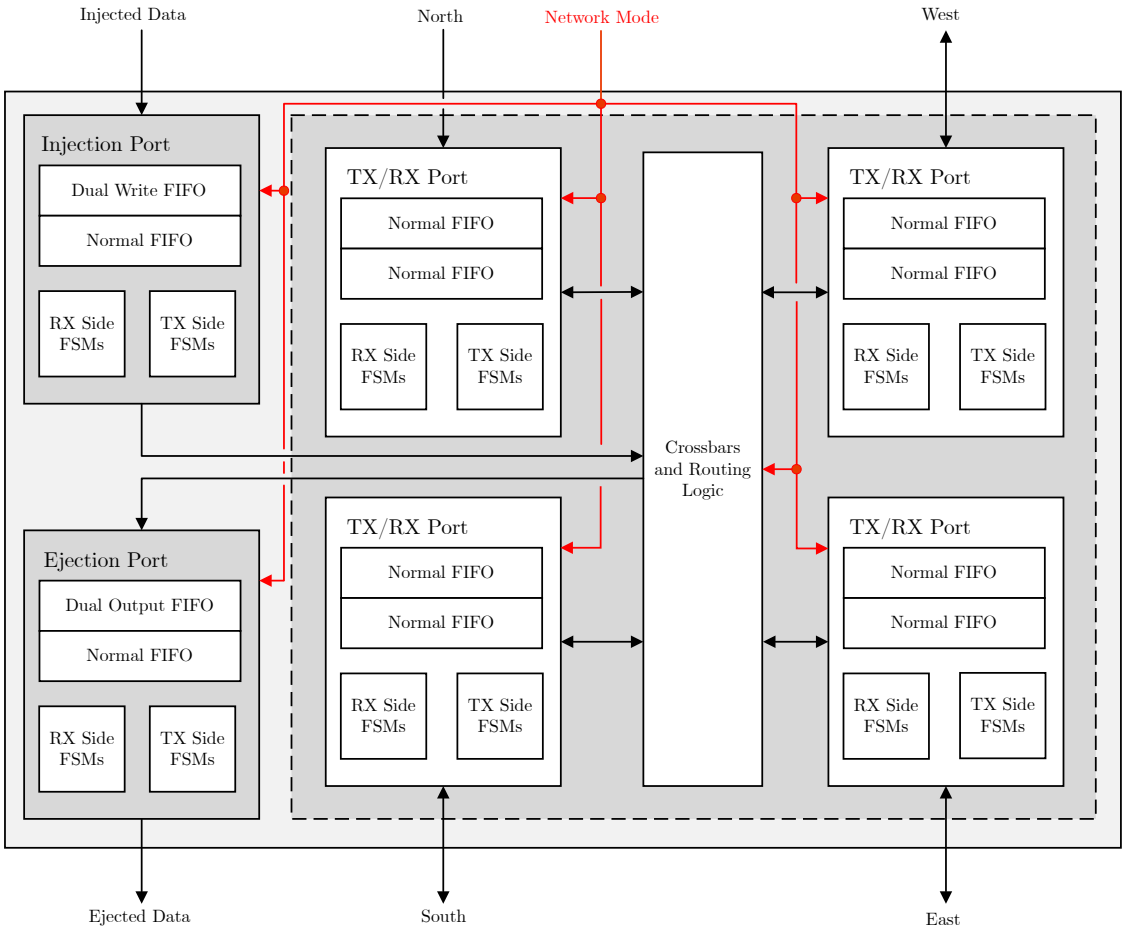


Figure 4.1: Block structure of a RELAX Mesh Tile.

4.2 Router

The router found in a RELAX tile is the collection of reception and transmission ports for the cardinal directions along with the crossbar. As well, the interfaces for the injection and ejection ports are provided. Structurally, the router is the same as shown as the black box in Figure 3.2. Each interface has a collection of data signals for parallel data transfer along with protocol signals as mentioned in Chapter 3. The high level representation of the router is shown as the dashed rectangle in Figure 4.1. If a port is not needed due to a router being on an extremity of the network, it is disabled by having the control signal and data inputs effectively zeroed out so that the router sees the port as never having any data available.

4.3 Buffers

4.3.1 Normal FIFOs

The most fundamental and used buffer in a RELAX tile is the normal FIFO. Each tile uses eight of these units, one per virtual channel across the four cardinal direction links. These buffers are traditional circular buffers, implying their read and write pointers wrap back to zero when incrementing from the last index [77]. As well, this buffer serves as the basis for the other specialized dual mode buffers used in the injection and ejection ports.

Structurally, the normal FIFO is implemented in a two-dimensional decoding scheme [78]. In particular, the buffer is composed of a $n \times 2$ array of registers, where n is the number of rows present. This architecture was chosen due to its ability to mitigate large decoding propagation delays, especially when these buffers can vary in size due to their parametric design. As well, this architecture lends itself to the specialized buffers where two adjacent cells may be manipulated in a single clock cycle. The structure can be visualized in Figure 4.2.

When reading from the buffer, all the bits in the read pointer except the least significant are used to select the row. In Figure 4.2, the read row select would be controlled by bit $\mathtt{B}_1$. The whole content of each row is passed to a row select multiplexer. Assuming a virtual channel size of N bits, this implies that each input to the multiplexer receives 2N bits of data. Lastly, bit $\mathtt{B}_0$ of the read pointer controls which N sized segment of the 2N string is output.

Writing to the buffer works in a similar mechanism to reading. A row select decoder is used in conjunction with all the bits of the write pointer aside from the least significant one. The least significant bit is used as a column select. The result of the column select ($\overline{\mathtt{B}_0}$ for column 0, $\mathtt{B}_0$ for column 1) and the decoder row output is fed through an AND gate per slot to act as a write enable.

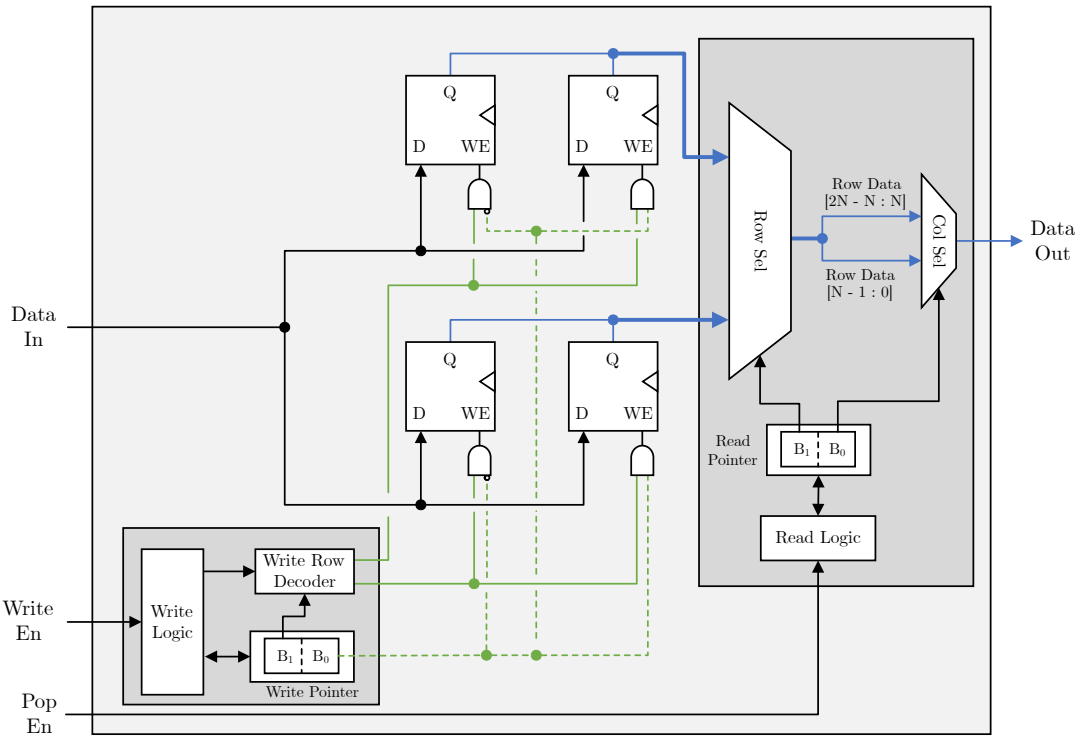


Figure 4.2: Two column structure of a regular FIFO. In this example, a four slot FIFO is shown.

4.3.2 Dual Write FIFO

The Dual Write FIFO buffer is found in the injection port of a RELAX network interface. It has two behaviours of operation that are selectable. When instanced in the injection port, the selected network mode will be used to determine which behaviour is desired. At the top level, the Dual Write FIFO has a data input port of 2N bits, where N is the specified FIFO width but has a data output of N bits. The RTL structure of this FIFO buffer can be seen in Figure 4.3.

Should the selected mode be set to zero, implying the network is operating in accurate only mode, this buffer will act as a normal FIFO described in the previous subsection. Though there are 2N bits as an input, only the least significant N bits are written into a single slot per clock cycle. The top-most bits are simply just ignored through the multiplexing action found at the D input. This implies that two writes are needed to proceed to the next row in the buffer's array of memory elements since the write pointer is incremented by one.

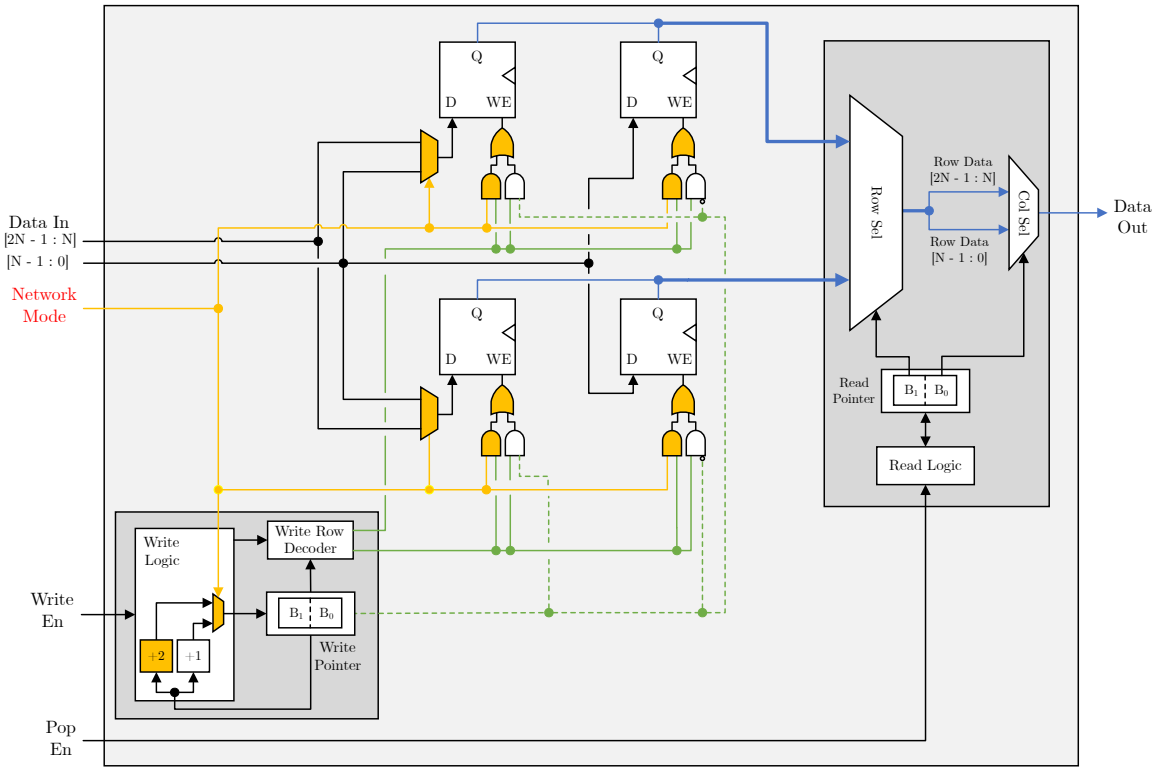


Figure 4.3: Simplified functional representation of a Dual Write FIFO. Additional added hardware such as logic gates and multiplexers needed to implement the dual write ability are shown in gold.

In order to provide the ability to write to two adjacent cells, the write row select is used in conjuction with an AND gate. The AND logic from the normal FIFO is also present. Both AND gates are fed into an OR gate as seen in Figure 4.3. When the buffer is set to operate in dual write mode, each write will store the entire string of 2N bits into an entire row of the buffer. Due to the toggling of the multiplexer, the write pointer will be incremented by two for each write operation. It should be noted, however, that the buffer will still eject one slot of N bits at a time.

4.3.3 Dual Output FIFO

The Dual Output FIFO is found in the ejection port of the network interface. It acts in an opposite manner of the Dual Write FIFO. This implies that the buffer has a data input of N bits but has an output 2N bits wide. Because of this, data is retrieved one row

at a time. Irrelevant of the chosen output size, the buffer will always output valid data in the lower N segment (Out_L). Out_H, the higher N bit segment, is provided by a chain of two multiplexers. The first (innermost) multiplexer selects the slot, while the second masks Out_H depending on the operating mode. The general structure of this module is shown in Figure 4.4.

If the dual output mode be disabled, the higher N bit segment (Out_H) will be masked and simply return a string of zeros provided by the final output multiplexer. As well, the read pointer value will have its value incremented by one due to the multiplexer feeding the pointer's input. If the dual output mode is enabled, Out_H will not be masked. As a result, a whole row of the buffer will be outputted. To account for this change in behaviour, each pop of the buffer will increment the read pointer by two and simply traverse rows, foregoing alternating columns.

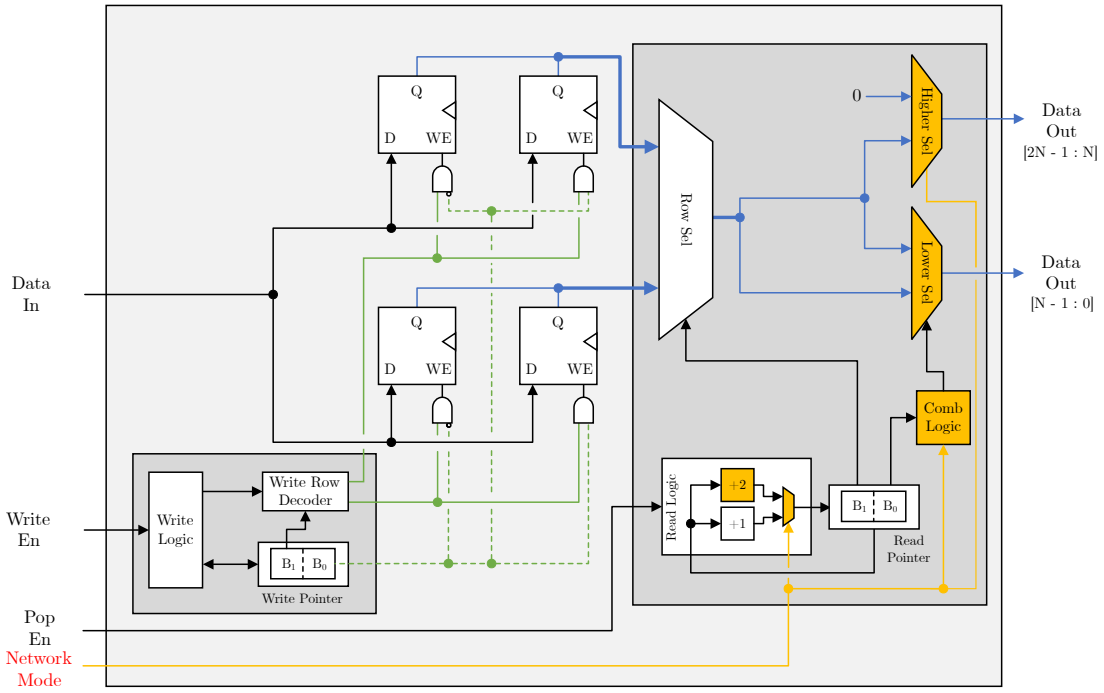


Figure 4.4: Simplified functional representation of a Dual Output FIFO. Additional added hardware needed for operating in Mixed Mode is shown in gold.

4.4 Reception and Transmission Ports

Within a mesh RELAX router, there are a total of four available full duplex links between routers. Each port contains two regular FIFO buffers, one for each virtual channel. As well, there are two pairs of TX and RX finite state machines instantiated to implement the RELAX packet transaction protocol discussed in Chapter 3. It should be noted that the network mode has no bearing on the functionality of these ports.

4.5 Crossbar and Routing Logic

The crossbar and routing block in RELAX is composed of several sub-components. Traditionally, a crossbar is composed entirely of combinational logic with the intent of passing data and control signals to the appropriate locations, while an arbiter will control said crossbar. With RELAX, this typical design is deviated from to account for the ability to change operating modes. Instead, two arbitration finite state machines are instantiated. In fact, the crossbar block found in a router contains the two arbitration FSMs and crossbars previously shown in Figure 3.6 along with additional combinational logic to implement the routing logic and strategy. Given a mesh network, there are five instantiations per outbound port in a RELAX router.

When operating in Accurate Mode, the single cycle FSM controls the data going to both buffers of the associated transmission port. This is achieved by having packet destinations fed to a multiplexer that is controlled by the FSM. The Mixed Mode FSM is effectively disabled and its outputs ignored. Whichever port is selected, its packet is placed on the transmission data ports in a single clock cycle through the multiplexing mechanism shown by Figure 4.5. Though not shown in the figure, status and control signals are passed from the RX port to the TX in a similar manner.

As can be also seen in Figure 4.5, if Mixed Mode is selected, this aforementioned FSM is delegated to handle Virtual Channel A and its approximate data. The second FSM is engaged and configured to move a packet to the transmission port in two clock cycles since the accurate packet has been serialized. Two combinational blocks are instantiated and

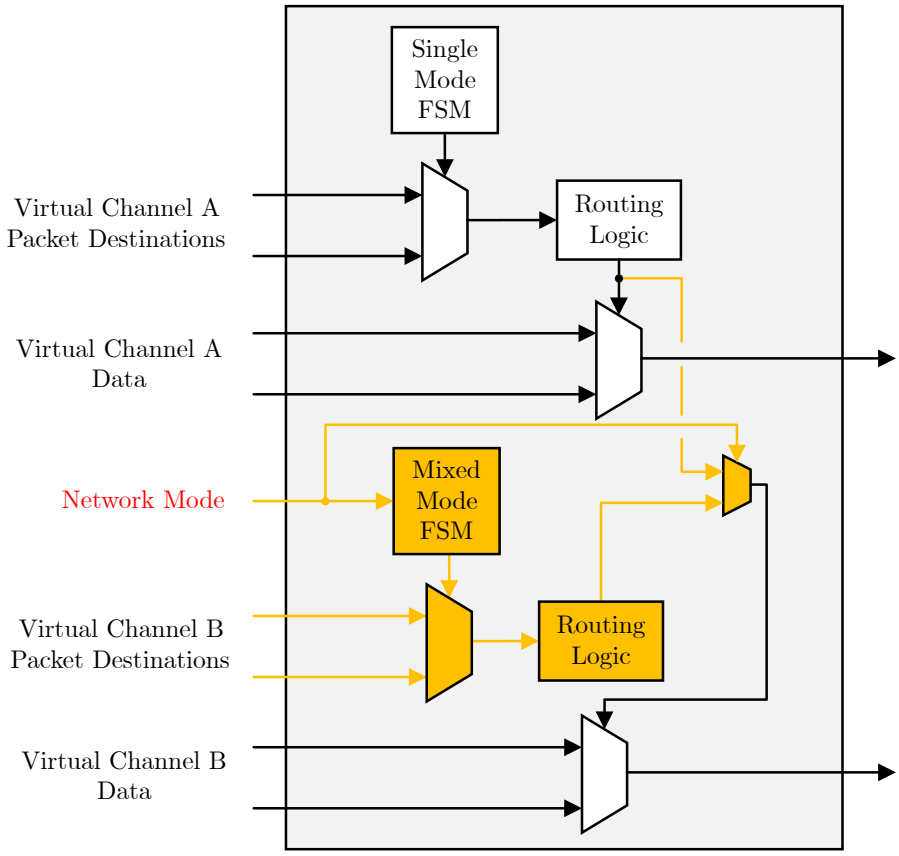


Figure 4.5: Crossbar and routing RTL structure for a single port. Added hardware to support Mixed Mode is shown in gold.

perform the actual routing, controlling the multiplexers of the data. A closer, in-depth look of the crossbar and routing control for a single transmission port can be seen in Figure 4.5.

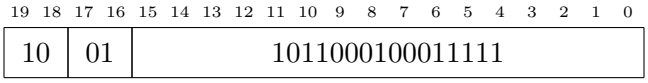
4.6 Flit Encoding, Ejection and Injection Ports

4.6.1 Flit Encoding

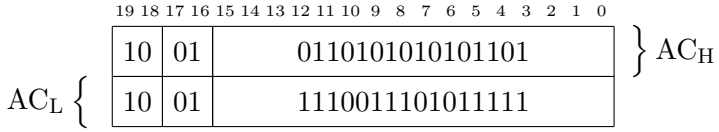
RELAX may transmit packets composed of one or two flits. One flit transfers are approximate packets, while two flit transfers are accurate packets. As previously mentioned, an accurate packet may be transferred as two flits in parallel if in Accurate Mode or in serial, should the network be running in Mixed Mode. A flit itself does not encode the type of data present within but rather is determined based solely on the virtual channel(s) used. All flits have the same structured used, as shown in Figure 4.6a with the destination encoded

in the most significant bits.

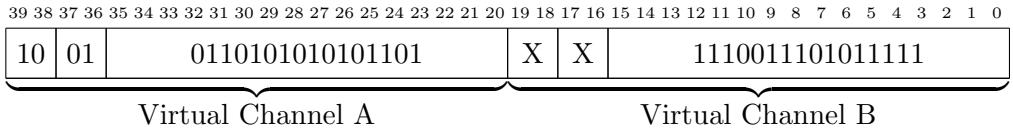
(a) Sample flit structure. The X and Y address fields are sized log_2(# of X addresses - 1) and log_2(# of Y addresses - 1) respectively. The payload is composed of N bits.



(b) Sample approximate flit going to X = 2, Y = 1 assuming a 2X2 mesh network, with a payload of `0b1011000100011111`.



(c) Sample accurate packet in Mixed Mode, destined to X = 2, Y = 1 assuming a 2X2 mesh network. The full payload is `0b01101010101011011110011101011111`. AC_H is transmitted first, followed by AC_L.



(d) Sample accurate packet in Accurate Mode, destined to X = 2, Y = 1 assuming a 2X2 mesh network. The full payload is `0b01101010101011011110011101011111`. Bits 16 to 19 are don't care values since the destination location is obtained from the flit in Virtual Channel A.

Figure 4.6: Flit structure and sample packets

4.6.2 Ejection Port

The ejection port is the point of exit for data in RELAX. It is composed of finite state machines, along with a normal FIFO attached to virtual channel A and a Dual Output FIFO connected to virtual channel B. The Dual Output FIFO is able to deserialize an accurate packet when running in Mixed Mode. Where the differences lies from other router link ports is the outputs of the port. The ejection port is designed to be connected to a processing element and has additional signals for account for this.

An attached processing element is given the status of the port with signals such if accurate or approximate data is available (assuming that Mixed Mode is enabled). If the processing element requests data, the same handshake protocol previously discussed is used

but the type of data is specified by asserting `dataType` to 1 if approximate data is desired. While RELAX supports concurrent transmission of both accurate and approximate data types, the link between an ejection port and processing element can only transmit one packet a time. As the packet is ejected, the origin of the packet is also output to the processing element as a string of bits sized log_2(# of tiles - 1). A black box view of the element can be seen in Figure 4.7a.

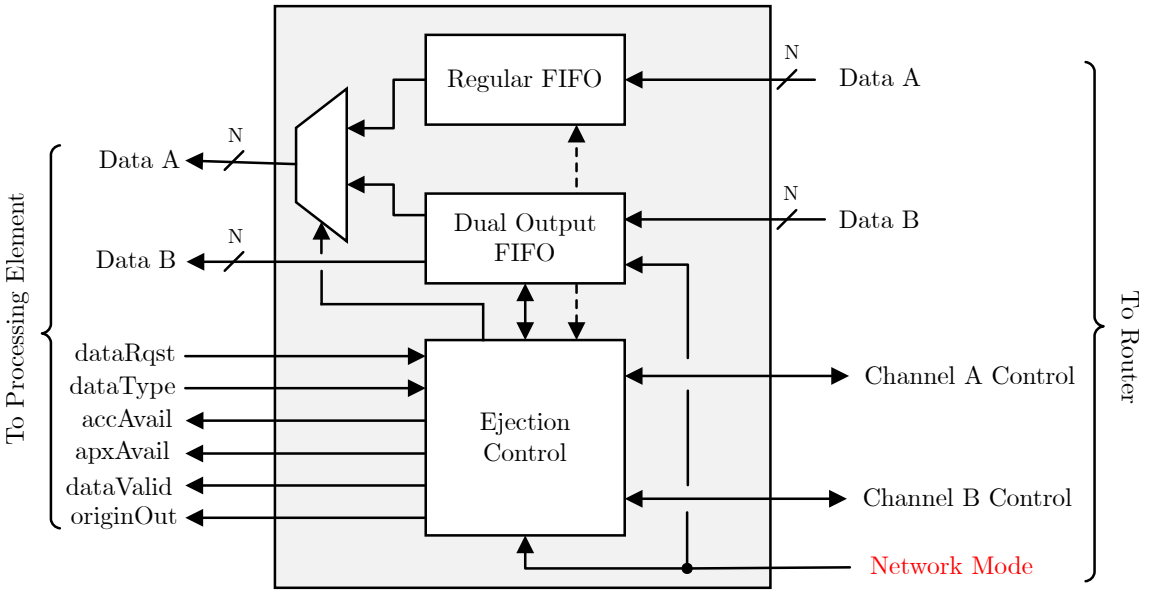
4.6.3 Injection Port

The injection port works in a similar manner as the ejection port. It acts as the point of entry for data onto the network. Just like the ejection port, it has finite state machines to control the buffers and a normal FIFO buffer tied to Virtual Channel A. However, a dual write FIFO buffer is used to serialize an accurate packet into two flits if the network is operating in Mixed Mode. Due to the structure of the Dual Write FIFO buffer, the serialization process only requires one clock cycle. An injecting processing element uses the same two signal handshake protocol to place data in the network. One packet may be placed at a time with the data type indicated by the `dataType` signal. When asserted, the data being injected is approximate. Figure 4.7b visualizes the connections between the injection port, processing element and router.

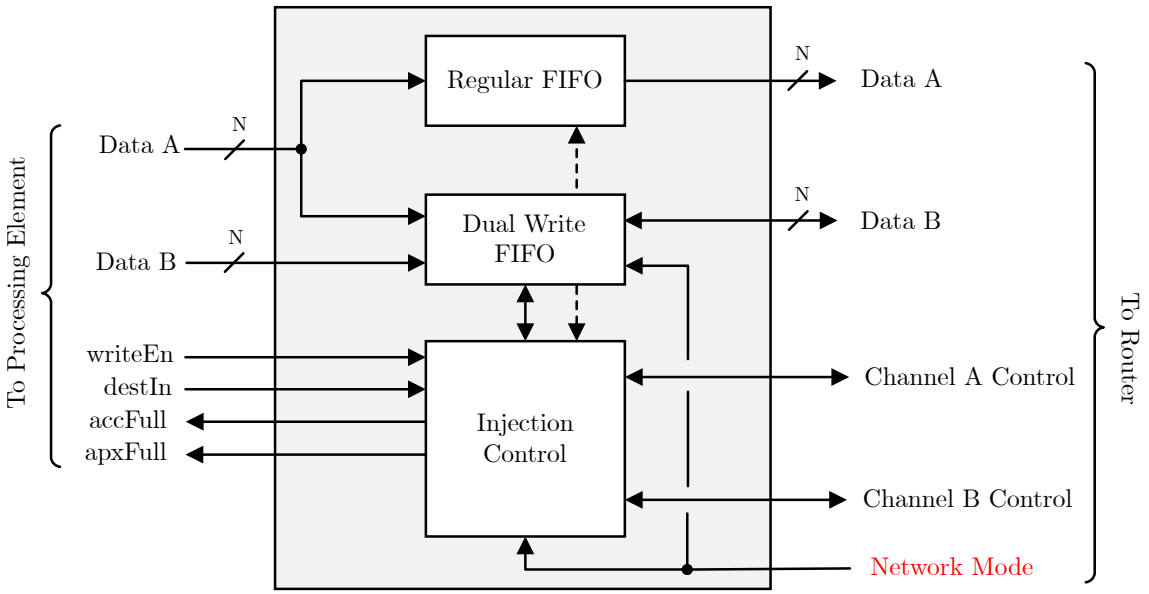
4.7 Summary

In this chapter, we presented the RTL design of RELAX. At the core of its design are three circular FIFO buffers: the normal FIFO, Dual Output FIFO and lastly, the Dual Write FIFO. The latter two FIFO buffers are critical to the network's ability of operating in Accurate or Mixed mode. As well, the two arbitration FSM structure of RELAX was discussed. The hardware overhead necessary to implement these features was analyzed and was found to be composed mostly of multiplexers along with some additional supporting logic.

Afterwards, the flit structure was reviewed. Using a single flit structure for transmission



(a) Ejection port black box, showing connections to processing elements and routers.



(b) Injection port black box, showing connections to processing elements and routers.

of accurate and approximate data in all cases allows for homogeneous hardware to be used, ultimately encouraging design reuse. Lastly, the mechanisms to inject and eject data from the network were analyzed. At the heart of packet injection is the Dual Write FIFO buffer, which allows for accurate packets to serialized in a single clock cycle, ensuring minimal latency. In terms of packet ejection, the dual output FIFO buffer is used to provide opposite functionality by deserializing data in a single cycle. With these two ports, a processing

element may inject or eject packets in a single cycle, irrespective of their data type. However, ejection and injection ports may handle only one packet at a time despite the rest of the network being able to transmit two packets concurrently.

Chapter 5

RTL Synthesis and Network Validation

5.1 Introduction

In this chapter, the performance and characteristics of RELAX are presented. First, synthesis is performed and as a result, associated attributes such as logic utilization and power consumption are quantified. Buffer components are synthesized for several widths and quantities of slots then compared for power and resource consumption. Afterwards, other network components such as routers and tiles are synthesized with a fixed virtual channel width of 22 bits then compared against the components from a baseline mesh network that handles solely accurate data. By doing so, an apples-to-apples comparison is achieved.

Lastly, network performance and validation is performed through a cycle accurate RTL simulation. The baseline network is tested with various injection rates, measuring latency and throughput. RELAX is subjected to the same testing conditions but with the addition of various accurate and approximate data distributions. The performance trends are then mapped according to well-established approximate computing tasks and their usage of approximate data.

5.2 Synthesis and Utilization

5.2.1 Environment

Each module is written in VHDL-2008 with synthesis performed by Xilinx Vivado ML 2021.2 with the Xilinx Virtex Ultrascale+ xcvu9p-fsgd2104 FPGA, running at speed grade 1 as a target. When performing synthesis, modules are provided with `KEEP_HIERARCHY` attributes to retain the inferred RTL hierarchy. Default strategies are used when performing synthesis and implementation/place and route, with no preference for performance or size reduction. Power consumption values of each module is determined by vector-less activity propagation analysis performed by Vivado with a clock of 100MHz.

5.2.2 FIFO Buffers

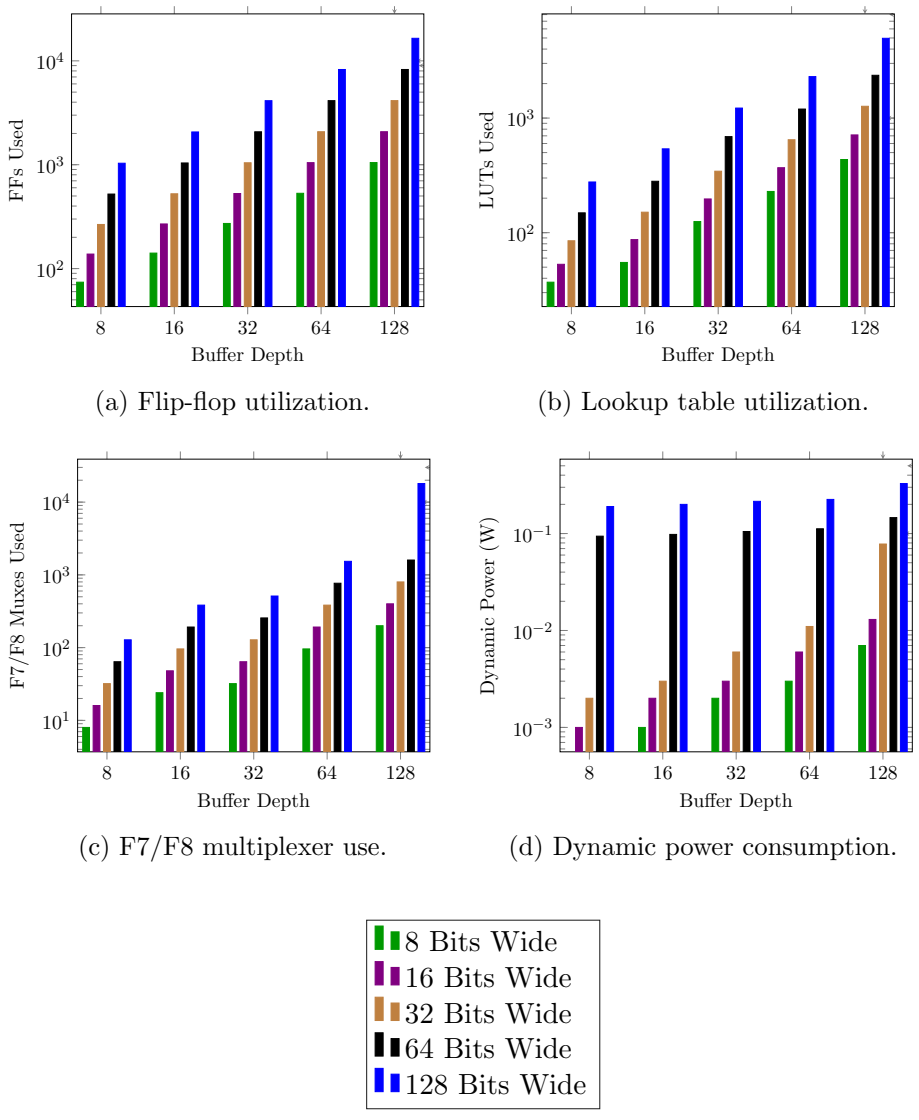
Scope of Testing

The three different FIFOs discussed in Chapters 3 and 4 are characterized by being synthesized for FIFO depths and widths of 8, 16, 32, 64 and 128 bits. Each buffer is synthesized 25 times to acquire results for each possible combination. The flip-flop and lookup table usages for each synthesis is plotted and then analyzed. Static power consumption is omitted from this study since it was found experimentally that the static power effectively remains consistent for a given width when varying depths.

Normal FIFO

When scaling the depth and width for normal FIFO buffers, it can be seen that the resource utilization increases linearly for flip flops and lookup tables. Seven and eight input multiplexer (F7/F8 mux) use, however, ramps up dramatically when a buffer with a width of 128 bits is used. The use of F7/F8 muxes has a benefit since it is a primitive [79] and is already implemented in hardware. By using these primitives, delay can be reduced since multiple lookup tables in a single slice do not need to be linked together to produce the same functionality. As well, using F7/F8 multiplexers does not incur any additional penalty in static power consumption since it is an internal component inside a logic slice.

This increased use can be attributed to the fact that there is now sufficiently wide data to facilitate use of these primitives. Overall, this is an acceptable result since it implies that instantiating larger FIFO buffers will incur stable, predictable increases in resource utilization with minimal increased power consumption.



(a) Flip-flop utilization.

(b) Lookup table utilization.

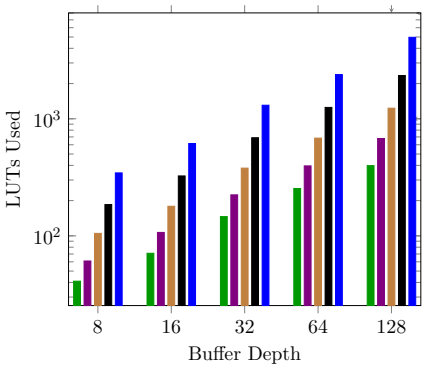
(c) F7/F8 multiplexer use.

(d) Dynamic power consumption.

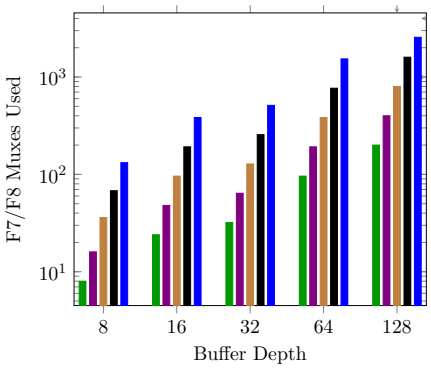
Figure 5.1: Logic utilization and power consumption characteristics of a normal FIFO buffer with differing FIFO buffer depths and widths.

Dual Write FIFO

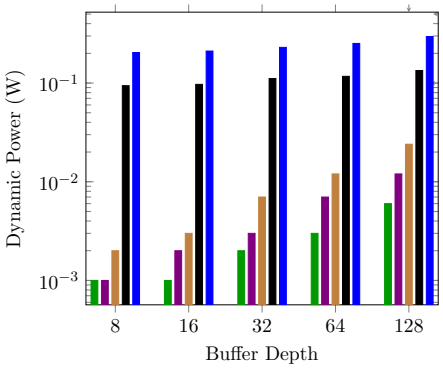
The Dual Write FIFO buffer exhibits nearly identical utilization characteristics as the normal FIFO and has linear trends for usage. The difference in resource usage when compared to a normal FIFO buffer is small and as a result, only the differences are plotted in Figure 5.2. Resource usage and power consumption grow until a certain threshold. Deducing from the utilization reports, there is a total of 386 more LUTs (+7.78%) used in the Dual Write buffer with a depth and width of 128 bits, but two less registers and 640 less F7/F8 multiplexers (-3.55%). Similar but less drastic trends are visible for other depths as well. It can then be assumed that the obtained reduction in power consumption, register and multiplexer use results from an implementation optimization, trading LUTs for other elements in order to yield better routing.



(a) Lookup table utilization.



(b) F7/F8 multiplexer use.



(c) Vectorless dynamic power consumption.

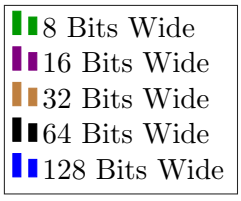
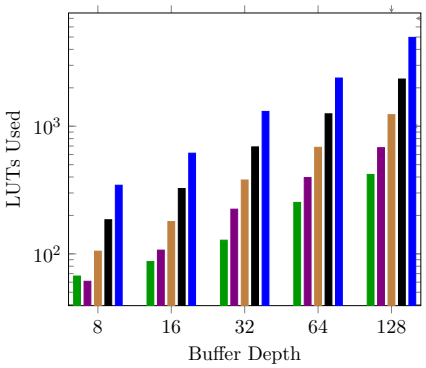


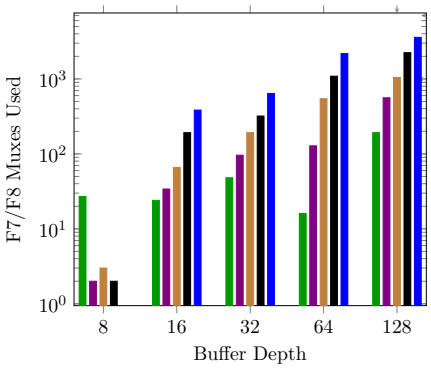
Figure 5.2: Logic utilization and power consumption characteristics of a Dual Write FIFO buffer compared to a normal FIFO buffer depths and widths.

Dual Output FIFO

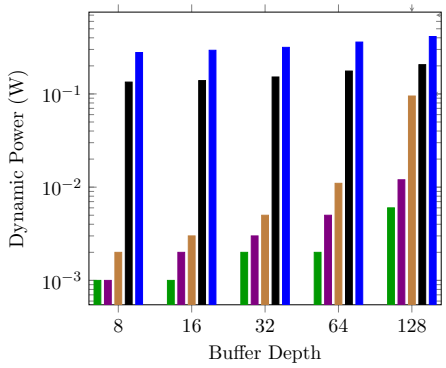
The Dual Output FIFO buffer has the largest cost to implement. The difference in lookup table use when compared to the normal FIFO buffer tends to increase in an exponential trend. Given the nature of having an optional ability to output two adjacent slots, large combinational paths may be inferred to multiplex outputs accordingly. Additionally, depending on the width of the buffer, these multiplexer paths may not necessarily benefit from the F7/F8 multiplexer primitives available on FPGAs due to not being sufficiently large or deep. Ultimately, as seen in Figure 5.3, deserializing and ejecting data is costlier than injecting data into the network.



(a) Lookup table utilization with differing buffer widths and depths.



(b) F7/F8 multiplexer use with differing FIFO buffer depths and widths.



(c) Vectorless dynamic power consumption.

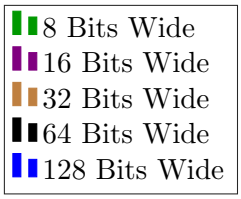


Figure 5.3: Logic utilization and power consumption characteristics of a Dual Output FIFO buffer compared to a normal FIFO.

5.2.3 Tile Component Resource Usage

The components of RELAX and baseline networks were synthesized, with resource usage for both the baseline and RELAX components being tabulated in Table 5.1. A FIFO buffer width of N = 22 was used, implying that RELAX has two virtual channel buffers with a payload of 22 bits wide, while the baseline has a single buffer with a 44 bit payload per link. Each scenario allocates 96 slots per buffer and assumes a 4X4 mesh network. While performing synthesis and place-and-route, f_{max} is found to be 175MHz for the RELAX network tile through static timing analysis.

The resource usage of each module for both baseline and RELAX versions can be seen in Table 5.1. TX and RX ports share the same resource usage in either baseline or RELAX variants. Any overhead incurred in the RELAX design of the TX and RX ports is attributed to the ability to operate the N bit wide virtual channels independently as opposed to having a single 2N bit wide channel. When comparing relative or percent-based differences, the crossbar and routing logic have the largest increase. This is due to the fact that when RELAX is operating in Mixed Mode, two crossbars and routing finite state machines are needed for the network to operate, effectively doubling the overhead. Injection and ejection port overheads are attributed to providing serialization and deserialization capabilities when operating in Mixed Mode. Overall, per tile, RELAX requires an approximate additional 45% increase in resources.

Module	Baseline		RELAX	
	LUTs ($\times 10^3$)	FFs ($\times 10^3$)	LUTs ($\times 10^3$)	FFs ($\times 10^3$)
TX Port	1.62	5.06	1.84 (**+13.6%**)	5.83 (**+15.2%**)
RX Port	1.62	5.06	1.84 (**+13.6%**)	5.83 (**+15.2%**)
Ejection Port	1.61	4.67	2.08 (**+29.2%**)	5.08 (**+8.78%**)
Injection Port	1.52	4.62	1.96 (**+29.0%**)	5.85 (**+26.6%**)
Crossbar	0.13	0.005	0.17 (**+30.8%**)	0.01 (**+100%**)
Router	13.3	40.5	14.8 (**+11.3%**)	46.6 (**+15.1%**)
Tile	13.0	39.7	18.8 (**+44.6%**)	57.5 (**+44.8%**)

Table 5.1: Resource utilization of RELAX components vs baseline.

As well, the power consumption of both baseline and RELAX tiles is determined with

vector-less propagation through the synthesis tool and a frequency of 100MHz for virtual channel widths of 8, 16, 32, 64 and 128 bits. For example, assuming N is equal to 8 bits, the baseline channel width is 2N or 16 bits. The results are tabulated in Table 5.2. RELAX's dynamic power consumption grows linearly with virtual channel width, producing an R^2 value of 0.9968 when plotted as a linear trend.

Likely due to place and route effort and the primitives available on the Virtex Ultrascale+ FPGAs, the best trade-off in terms of power consumption is operating with a 128 bit wide virtual channel. Static power consumption is subjected to a minor increase of 2mW (approximately 0.08%), while dynamic power increases 0.189W or 26.7%. This is likely due to the synthesis tool being able to effectively use larger primitives instead of inferring equivalent logic for smaller channel widths. A more accurate dynamic power consumption estimation would be obtained through performing an RTL simulation then using a `.saif` vector file generated as a by-product.

Virtual Channel Width (N)	Baseline Static (W)	RELAX Static (W)	Baseline Dynamic (W)	RELAX Dynamic (W)	Δ Dynamic (W)
8	2.470	2.469	0.082	0.143	0.061 (+**74%**)
16	2.470	2.471	0.131	0.169	0.038 (+**29%**)
32	2.471	2.472	0.193	0.293	0.100 (+**51.8%**)
64	2.473	2.475	0.362	0.476	0.114 (+**31.5%**)
128	2.478	2.480	0.707	0.896	0.189 (+**26.7%**)

Table 5.2: Power consumption of a RELAX tile compared to the baseline for various channel widths at a frequency of 100MHz.

5.3 Simulated Testing Environment and Flow

5.3.1 Testing Overview

A 4X4 top-level mesh network consisting of RELAX tiles operating in Mixed Mode are set to inject packets randomly with arbitrary destinations. The baseline test is RELAX operating in Accurate Mode. Using the implementations written in VHDL-2008, the network

is simulated using Mentor Graphics Modelsim for various injection rates and distributions of accurate and approximate packets. The randomization functions are provided by OSVVM and are used to determine when a packet is to be injected, the destination of said packet and the packet type (accurate or approximate).

5.3.2 The Case for Controlled Randomization

When testing digital designs, random stimuli are used for testbenches and verification. However, a case can be made that reproduceable randomization where the same results are obtained if the same seeds and other inputs are used is desirable [80]. The justification for this scenario is that when verifying a design, using the same stimuli or inputs is necessary to aid in validation especially if a failed test case needs to be reproduced.

OSVVM's randomization functions are designed with this reproduceable property in mind. Given that network performance metrics such as latency and throughput are being tested; true, non-reproduceable randomization is desired to account for stochastic events such as operating system context switches affecting when packets may be injected. The RandomPkg of OSVVM has a defined type called `RandomPType` that is a reproduceable randomizer that requires a seed input for initialization. To ensure that each network tile does have true randomized injections, the randomization seed for a given network tile contains the instance name (which differs tile from tile) and a supplemental Mersenne Twister [81] value returned from Python's `random()` function in the range of [0.0, 1.0).

5.3.3 Testing Environment

The testing environment is composed of two components: a Python pre- and post-processing script and a Modelsim simulation. Upon launching the Python script, the injection rates and data type distributions are determined from a configuration file. Simulations are divided into sets, consisting of one data type distribution tested against all injection rates. A simulation pass is performed in three stages:

(1) Pre-processing

The network configuration file, `noc_parameterspkg.vhd`, is updated by the Python script to include the new data type distribution and injection rate. As well, the injection driver is updated to include an updated seed including the aforementioned `random()` value.

(2) RTL Simulation

The Python script then invokes Modelsim, which compiles all the VHDL files with updated values then launches the simulation. During the operation of the simulation, a `.csv` file is produced per tile with each line corresponding to a received packet. On each line, the packet's origin, injection and ejection times and type are recorded.

(3) Post-processing

Once Modelsim has exited, the Python script then combines all the generated `.csv` files into a single file located outside of the RTL simulation's working directory.

Once a set of simulation passes is completed, the Python script will read all `.csv` files of that set and place the data in an Excel sheet for ease of access and presentation. The working directory is then sterilized of all `.csv` files and the process is repeated until no there are no more sets to test.

5.3.4 Testbench Flow

Overview

For testing purposes, a testbench written in VHDL-2008 is designed to instantiate a 4X4 mesh network. At the core of this testbench are two timekeeping registers along with non-synthesizable injection and ejection drivers that attach to the interfaces of their respective ports.

Timekeeping Registers

The first timekeeping register keeps track of the simulation time, or rather, how many simulation cycles has passed. This information is used to determine the duration of a generated packet's travel time. When a packet is injected, the current simulation time value is encoded into the flit. The total travel time is the difference between when the ejection port ejects the packet and the encoded injection time. This value is written in the recipient tile's corresponding `.csv` file when ejecting the packet.

Execution of injecting a packet depends on the second timekeeping register known as the injection period counter. A property known as the injection period is defined in the network parameters file. Its purpose is to count the number of cycles, incrementing alongside the timekeeping register, however rolling over after exceeding the number of cycles specified in the parameters file. As a result, when performing a simulation pass, the injection rate is defined as

$$\text{Injection Rate} = \frac{\text{Number of packets to inject per cycle}}{\text{Injection period size}} \tag{1}$$

with the number of packets to inject per cycle defined in the parameters file as well. This value, though, is generated then written in by the Python preprocessing segment.

The Injection Driver

Before the simulation may begin, a reset must be issued. During this reset, each injection driver generates a vector of times within the injection period to push a packet into the network. These times are produced by using the randomization package of OSVVM and fed into a data type known as an integer vector (VHDL-2008 standard data type). Given the randomization context mentioned previously, the injection driver of each tile produces a different set of values from having a seed composed of the tile's instantiation name in the simulation, its coordinates in the network and the random value produced by Python's `random()` function. The injection vector generation is implemented by the following VHDL process:

```
-- Generate injection times
times_gen_proc : process (clk, rst) is
    variable injectionTimes_rand : randomptype;
begin
    if (rst = '1') then
        report "Generating New Seeds"
            severity note;
        injectionTimes_rand.InitSeed(injectionTimes_rand'instance_name &
            to_string(x_coord)&to_string(y_coord) & "0.7225013680664656");
        injectionTimes_i <= injectiontimes_rand.RandIntV(0, (packet_period_size - 1),
            packets_per_period, packets_per_period);
    end if;
end process times_gen_proc;
```

As seen in the code snippet, the value `0.7225013680664656` is the value produced by `random()` during preprocessing. Each pass of the simulation will produce a different value. As a result, there is substantial insurance that each tile will have notably varied injection times within the injection period.

The `RandIntV()` function produces a randomized integer vector with the following parameters in order as shown in the code snippet:

(1) Smallest value

In this case, start at zero to maximize the range of available values.

(2) Largest value

Similarly, end at `packet_period_size - 1` to maximize the range of possible values.

(3) Number of values

Use the value generated in preprocessing with the injection rate and then specified in the parameters file.

(4) Number of unique values

All values should be unique (ie, no injection of two packets at once by one injection driver) so the number of values desired is also the number of unique values to produce.

Inside each injection driver, there are two additional components of note. First is a buffer for packets to be injected. This exists to handle any potential backpressure from the latency when injecting accurate packets which require two cycles to fully propagate through and is sized rather large. The second component is a counter for the number of packets injected. Having the counter ensures that the injection driver does not exceed the number of packets required to be injected. For all tests documented in this chapter, the value is set to 1000, implying a total of 16000 packets are generated and injected throughout the network per pass.

Injection Flow

During runtime and on each positive clock edge, an injection process is triggered. The injection driver first checks to see if there is a value inside its injection times vector that matches the current injection period counter value. If there is a match and the buffer is not full, the injected packet counter is incremented and then a packet is generated and to be injected. The data type and destination are generated randomly by once again using OSVVM.

A Boolean value is generated from a randomization function that uses a distribution of probabilities specified in the parameters file. If the resulting value is zero, the packet is accurate. A result of one will produce an approximate packet. This distribution is defined as `apx_weight` and `acc_weight` in the parameters file and dictates the distribution used to determine if a packet will be accurate or approximate should the simulation operate in Mixed Mode. As an example, if `apx_weight` is set to 60 and `acc_weight` is set to 40, there is a 40% chance that a generated packet will be accurate. The cycle that generated the packet is encoded into the flit and placed into the buffer for injection. Figure 5.4 displays the process in a flowchart. This process is used to generate artificial traffic used to characterize the performance of the network.

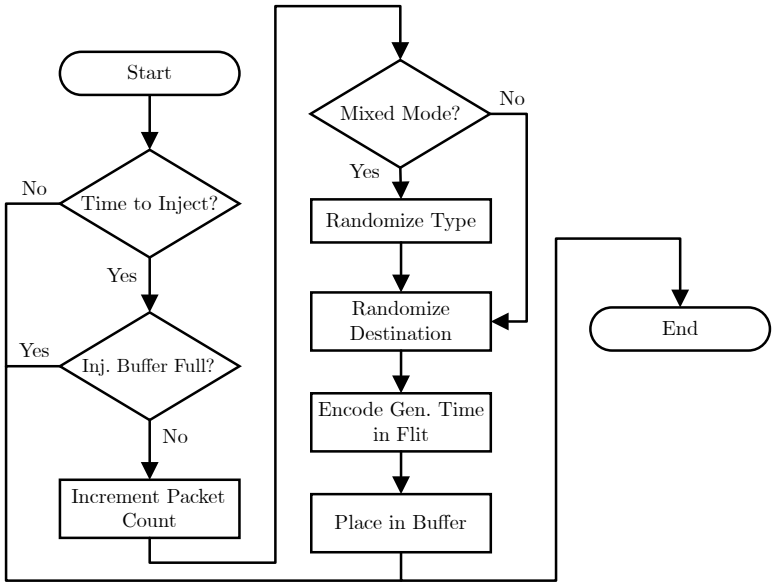


Figure 5.4: Flowchart of packet injection process.

5.4 Network Performance Metrics

5.4.1 Testing Methodology and Sampling

While beneficial, the randomization methodology used does not provide exact, reproduceable results. As a result, the quantity of approximate and accurate packets can vary since the packet data type is determined by a distribution of probabilities. When plotting results such as latencies, this randomization can be present in the form of ripples in the plots. To account for this, the latency and throughput metrics presented in this section are the result of averaging 30 runs of each simulation pass in order to smooth out the plot to an acceptable degree.

5.4.2 Network Latency

The network latency is measured by using various distributions of approximate and accurate data tested across a range of injection rates from 0.01 to 0.30. It is computed as the average number of cycles needed for a packet to reach its destination once it has been injected into the network while accounting for the distribution used. As well, the baseline case of accurate only data is considered in this series of tests and is implemented as the

network operating in Accurate Mode as opposed to using a separate implementation.

The results are plotted in Figure 5.5. It can be seen that the case of having only approximate data being transmitted is functionally equivalent to the baseline case (one flit per packet). For such traffic, virtual channel B remains vacant. As a result, the latency penalty from having two flits is avoided. A ratio of 75% approximate data and 25% accurate data would lead to an overall latency improvement, which is realized from the relative increase in the injection rate. The best-case scenario is found to be when approximate traffic accounts for two thirds of the total traffic. This distribution allows to evenly utilize RELAX's channels. For traffic involving less than 50% of approximate data, we observe a significant increase in the latency due to contention in the injection. This issue can be overcome by configuring RELAX to Accurate Mode.

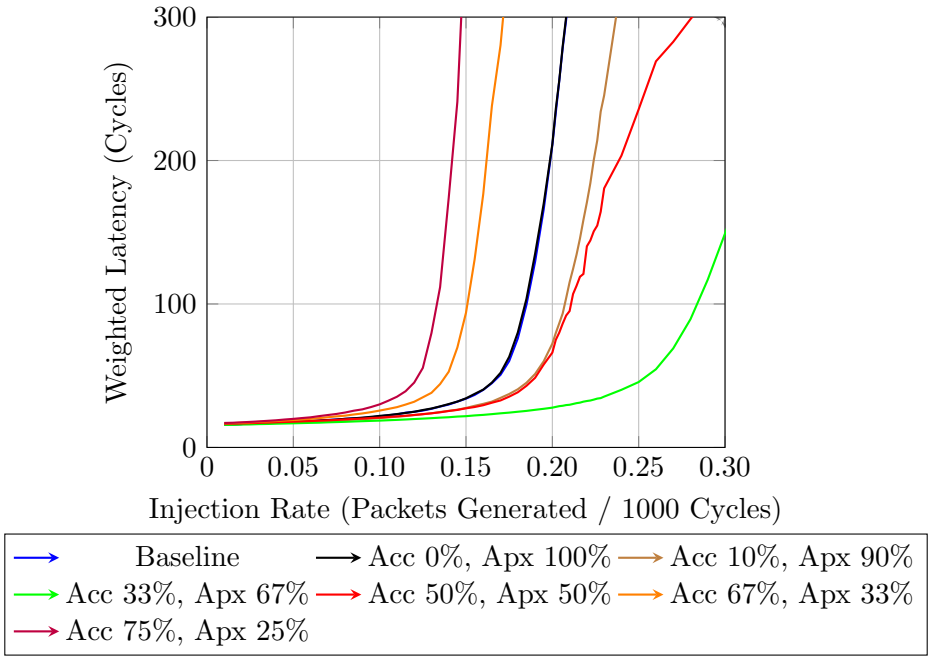


Figure 5.5: Weighted latency across different injection rates.

5.4.3 Network Throughput

Throughput trends were computed by determining the number of packets that were injected then received within a predetermined sample window from simulation cycle 1000 to 5000. This window was selected as it was found that the network was sufficiently saturated

by analyzing the transmission transcript `.csv` files. Assuming a clock frequency of 100MHz, the throughput is obtained by using the averaged count of packets received in the sample window, divided by both the window size and frequency. The equation can be written as:

$$\text{Throughput} = \frac{\text{(Averaged) Packet Count}}{\text{Sample Window Size} \times \text{Frequency}} \tag{2}$$

Assuming a frequency of 100MHz, a sample window of 4000 cycles and a count of 1280.2, the throughput can be calculated:

$$32.005 \times 10^6 \text{ packets/sec} = \frac{1280.2}{4000 \times 100\text{MHz}} \tag{3}$$

The results are plotted in Figure 5.6. Out of the distributions plotted, the best case scenario is when there is a 10% probability of accurate data (+6.6%) and the worst being a 75% probability of accurate data (-23.5%). These results are due to the fact that an approximate packet requires the transmission of one flit versus the two needed for accurate data in the mixed mode, resulting in an additive latency penalty for accurate data in Mixed Mode. Consequentially, the throughput is inversely affected by the additional latency. That said, based on the results presented in Figure 5.5, it can be deduced that once accurate packets account for more than 50% of the total network traffic, the network performs worse than baseline. In these situations, it would be advantageous to operate the network in Accurate Mode.

5.4.4 Suitable Use Cases

In order to leverage RELAX's performance gains, workloads must be selected based on the distribution of approximate and accurate data. We consider potential applications that have highly capable approximate workloads to evaluate the efficacy of RELAX. First, a well known pair of approximate benchmarks, ConvTex and DCT, are compared against RELAX. These benchmarks are able to yield a distribution of over 85% normalized approximate traffic [63].

Given a distribution of 67% approximate traffic along with an injection rate of 0.50,

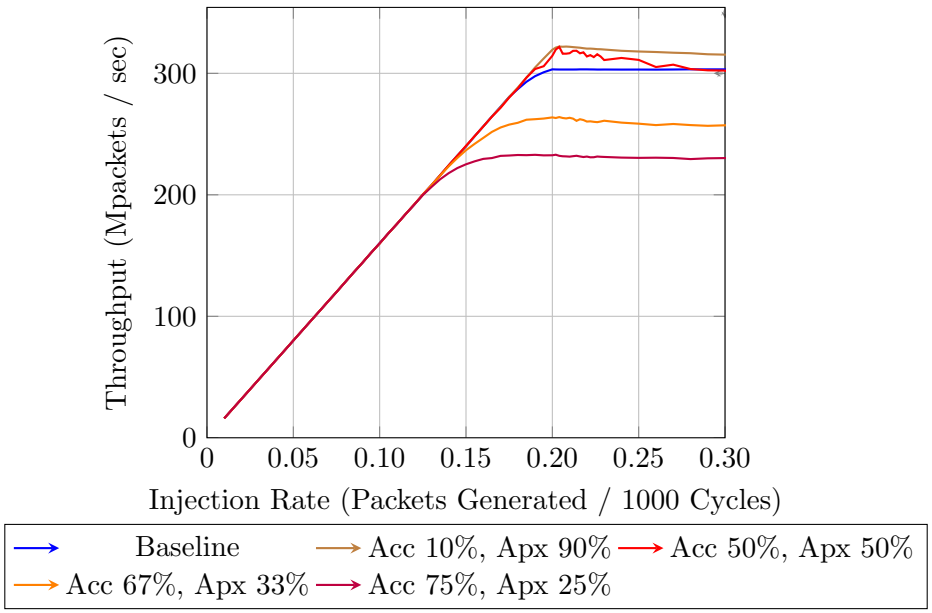


Figure 5.6: Throughput measurements when operating in the mixed data mode at a given frequency of 100MHz.

RELAX would be able to yield a 44.2% latency improvement when compared to a baseline. Latency improvements are only obtained when approximate traffic accounts for at least half of total network traffic. These gains increase until approximate traffic accounts for 2/3 of all traffic then taper back down until approximate traffic accounts for 100% of total traffic. In this situation, there is no latency gain to be had.

Chen and Louri determined in [25], that the blackscholes,fft, inversek2j, jmeint, jpeg and k-means benchmark applications of the AxBench benchmarking suite [26] yield 20% to 35% of global traffic as approximate and are considered to yield low volumes of approximate traffic across the network. These synthetic benchmarks may not necessarily produce ideal operating conditions to obtain the benefits that RELAX can provide.

5.4.5 System Level Simulations

While these tests do provide some characterization of network performance, executing applications or benchmarks is not realized. The results presented in Figures 5.5 and 5.6 represent probabilistic distributions of network traffic in a continuous stream during an RTL simulation. However, applications may have unique ebbs and flow in terms of computation

resource utilization. As a result, the view provided is a narrow perspective. To enrich this viewpoint, system level simulations may provide better optics in terms of how the proposed NoC handles application workloads involving reduced precision floating point numbers. Overall application speedups can be obtained and quantified rather than correlating network traffic distributions with benchmarks. The applications of interest are blackscholes; fft; inversek2j; jmeint; jpeg and k-means benchmarks of AxBench [26], along with discrete cosine transform and ConvTex benchmarks [63].

To facilitate these simulations, the gem5 simulator [82] would be an ideal candidate. gem5 is a well-established system level simulator that can execute binaries through Linux emulation, reducing the time to produce viable simulation results. By being able to execute these binaries, benchmarks such as AxBench can be ran at a minimal overhead cost. With that said, executing system level simulations is by no means a trivial endeavour and would be divided into three tasks.

The first task would be to modify gem5 to support precision scaling. One open source approximation fork of gem5 of interest is gem5-approxilyzer [83]. gem5-approxilyzer is a profiling suite for approximate performance and injects bit flips to quantify and characterize error resilience of applications. It does not provide any mechanisms for precision reduction approximation. As a result, the CPU models of gem5 would need to be modified to add this functionality.

Once the modifications to the CPU model have been performed, the compiler must be modified to incorporate and support these changes. The llvm [84] backend would be used to provide compilation duties as discussed in [67], where regions of interest are flagged to designate sections in which the processor core executing the code may approximate data. In these regions, the llvm compiler would use instructions created as part of the modified gem5 CPU models. As a result, benchmark code would need to be modified to indicate these regions of interest. It is at this point as well, a framework such as OpenMP would be needed to provide multithreading support for these benchmarks.

Lastly, the router and interconnects of the proposed NoC need to be implemented. In this context, Garnet [85], the interconnection framework provided as part of gem5 would

need to be extended to support RELAX. However, Garnet does not support variable bandwidth allocation. As a result, there is a major limitation present which would require substantial refactoring in order to support the bandwidth splitting action of RELAX before implementing the network interface.

5.5 Closing Thoughts

In closing, RELAX can yield notable performance improvements if appropriate workloads and network dimensions are chosen. In terms of physical cost, the best case scenario relative to the baseline is obtained with larger buffers and a large (virtual) channel width. To optimize for latency, a workload with 33% accurate data would be ideal. Contextually, the accurate data would be used for control and instructions while the approximate virtual channel would be used for passing the data to be manipulated.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis, we propose RELAX, a REconfigurabLe ApproXimate network on chip with a focus on the ability to transmit accurate and approximate data concurrently while facilitating resource sharing. At the core of RELAX is a mesh network topology with two virtual channels per link. Links can be configured globally to either facilitate the transmission of accurate data only using the entire available bandwidth, or split in two virtual channels to allow for concurrent transmission of approximate and accurate data simultaneously. By leveraging virtual channels, RELAX can allocate the entire link depending on the desired use case. It should be noted that RELAX itself does not perform any approximation but rather serves as the transport layer for data approximated by precision scaling and/or bit field reduction.

Previous works involving approximate networks on chip allow for data transmission of both accurate/protected data and approximate data, there is typically some limitation involving the transmission of both data types. In some cases, the network itself does not allow the concurrent transmission of both accurate and approximate data on a link [61], [62], [64]. With other designs, concurrent transmission of both types is possible, however resource utilization is high, and as a result, power consumption is elevated due to having two subnetworks operate [68] individually. This also implies that if the network is operating in

such a manner that only accurate or protected data is being transmitted, a sizable amount of the area and resources that are allocated for the design remain dormant and unused.

Through RTL simulations, it was determined that the operation of RELAX in Mixed Mode can have a positive impact on the latency so long as approximate packets account for more than 50% of network traffic. When operating in these conditions, the case scenario is approximate traffic accounting for two thirds of the total traffic on the network. Throughput benefits are maximized when the distribution of network traffic is 90% approximate. These network performance benefits can be obtained with an average 44.4% increase in lookup tables and a 44.8% increase in flip flops per tile when implemented on a Xilinx Virtex Ultrascale+ FPGA. For a given virtual channel width of 128 bits, these performance benefits can be realized with a 26.7% increase in power consumption.

6.2 Future Works

While RELAX produced promising results for network latency and throughput, these results were obtained in a blind manner through cycle accurate RTL simulation. Further profiling with more higher level simulations such as gem5 [82] with Garnet [85] or NOXIM [86] could provide better guidance in terms of where RELAX could be used more effectively. As well, having a RELAX implementation in gem5 with Garnet would allow direct profiling of network performance when running standardized benchmarks such as AxBench [26] without the need of matching latency and throughput trends to pre-established research.

As well, the power consumption estimates are certainly on the higher end of values due to having RELAX being instantiated on an FPGA. By implementing the network tile on an ASIC, major power consumption reductions can be achieved. Though omitted, static power consumption would benefit the most from the migration to an ASIC due to having to power configurable logic blocks.

Lastly, when selecting the operating mode of RELAX, the selection is global throughout the whole network. There may be cases where having solely accurate data in some portions of the network may prove to be beneficial, while other portions may require the

use of approximate data. Regions that need solely accurate data would not be subjected to the latency penalties when accurate traffic accounts for the majority, if not all, traffic in Mixed Mode operation as shown in Figure 5.5. This functionality could be implemented as a partially reconfigurable feature [87] and further extended by increasing the number of available virtual channels along with how the bandwidth is distributed among them. With a per-tile configuration, the number of use cases where RELAX can offer performance benefits is enlarged while potentially reducing hotspots [88] [89] within the network.

Bibliography

[1] E. P. Debenedictis, "It's Time to Redefine Moore's Law Again," *Computer*, vol. 50, no. 2, pp. 72–75, 2017.

[2] D. Hisamoto, L. Wen-Chin, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, K. Tsu-Jae, J. Bokor, and H. Chenming, "FinFET - A Self-Aligned Double-Gate MOSFET Scalable to 20 nm," *IEEE Transactions on Electron Devices*, vol. 47, no. 12, pp. 2320–2325, 2000.

[3] S. C. Song, B. Colombeau, M. Bauer, V. Moroz, X. W. Lin, P. Asenov, D. Sherlekar, M. Choi, J. Huang, B. Cheng, C. Chidambaram, and S. Natarajan, "2nm Node: Benchmarking FinFET vs Nano-Slab Transistor Architectures for Artificial Intelligence and Next Gen Smart Mobile Devices," in *2019 Symposium on VLSI Technology*, pp. T206–T207, 2019.

[4] S. S. Chung, C. K. Chiang, H. Pai, E. R. Hsieh, and J. C. Guo, "The Extension of the FinFET Generation Towards Sub-3nm: The Strategy and Guidelines," in *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, pp. 15–17, 2022.

[5] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys*, vol. 48, no. 4, pp. 1–33, 2016.

[6] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-Purpose Code Acceleration with Limited-Precision

Analog Computation," in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, IEEE, 2014.

[7] E. S. Dunstone, "Image Processing Using An Image Approximation Neural Network," in *Proceedings of 1st International Conference on Image Processing*, vol. 3, pp. 912–916 vol.3, 1994.

[8] S. Ataei and J. E. Stine, "A 64 kB Approximate SRAM Architecture for Low-Power Video Applications," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 10–13, 2018.

[9] M. E. Elbtity, H.-W. Son, D.-Y. Lee, and H. Kim, "High Speed, Approximate Arithmetic Based Convolutional Neural Network Accelerator," in *2020 International SoC Design Conference (ISOCC)*, pp. 71–72, 2020.

[10] S. Sen, S. Venkataramani, and A. Raghunathan, "Approximate Computing For Spiking Neural Networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 193–198, 2017.

[11] T. Yeh, P. Faloutsos, M. Ercegovac, S. Patel, and G. Reinman, "The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration," in *40th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 2007.

[12] M. Vilim, H. Duwe, and R. Kumar, "Approximate Bitcoin Mining," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: approximate data types for safe and general low-power computation," 2011.

[14] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and Compiler support For Auto-Tuning Variable-Accuracy Algorithms," in *International Symposium on Code Generation and Optimization (CGO 2011)*, pp. 85–96, 2011.

[15] Microsoft, "Microsoft Announces Release of DirectX 8.0," 2000.

[16] M. Macedonia, "The GPU Enters Computing's Mainstream," *Computer*, vol. 36, no. 10, pp. 106–108, 2003.

[17] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[18] V. Gupta, T. Li, and P. Gupta, "Lac: Learned approximate computing," *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1169–1172, 2022.

[19] E. Kilgariff and R. Fernando, "The GeForce 6 Series GPU Architecture," in *GPU Gems 2*, nVidia, 2005.

[20] M. Toksvig, P. Sriram, J. Matheson, B. Cabral, and B. Smith, "NVIDIA Tegra," in *2008 IEEE Hot Chips 20 Symposium (HCS)*, pp. 1–28, 2008.

[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[22] A. Das, A. Kumar, J. Jose, and M. Palesi, "Revising NoC in Future Multicore-Based Consumer Electronics for Performance," *IEEE Consumer Electronics Magazine*, vol. 11, no. 3, pp. 79–86, 2022.

[23] M. Ramirez, M. Daneshtalab, J. Plosila, and P. Liljeberg, "NoC-AXI interface for FPGA-based MPSoC platforms," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 479–480, 2012.

[24] S. Davidson, S. Xie, C. Torng, K. Al-Hawai, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor, "The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips," *IEEE Micro*, vol. 38, no. 2, pp. 30–41, 2018.

[25] Y. Chen and A. Louri, "Learning-Based Quality Management for Approximate Communication in Network-on-Chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3724–3735, 2020.

[26] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2017.

[27] R. Fenster and S. Le Beux, "RELAX: a REconfigurabLe Approximate Network-on-Chip," in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 381–387, 2021.

[28] R. V. K. Pillai, D. Al-Khalili, and A. J. Al-Khalili, "Power implications of precision limited arithmetic in floating point FIR filters," in *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, pp. 165–168 vol.1, 1999.

[29] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," in *2011 24th Internatioal Conference on VLSI Design*, pp. 346–351, 2011.

[30] A. B. Kahng and S. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs," in *DAC Design Automation Conference*, pp. 820–825, 2012.

[31] M. Masadeh, O. Hasan, and S. Tahar, "Input-conscious approximate multiply-accumulate (mac) unit for energy-efficiency," *IEEE Access*, vol. 7, pp. 147129–147142, 2019.

[32] IEEE, "IEEE Standard for Floating-Point Arithmetic," 2019 2019.

[33] N. Brunie, F. de Dinechin, and B. de Dinechin, "A Mixed-Precision Fused Multiply and Add," in *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pp. 165–169, 2011.

[34] S. Mach, D. Rossi, G. Tagliavini, A. Marongiu, and L. Benini, "A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018.

[35] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Invited: Approximate Computing With Partially Unreliable Dynamic Random Access Memory," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–4, 2016.

[36] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[37] W. Penny, G. Correa, L. Agostini, D. Palomino, M. Porto, G. Nazar, and B. Zatt, "Low-Power and Memory-Aware Approximate Hardware Architecture for Fractional Motion Estimation Interpolation on HEVC," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[38] G. Duan, "Research on Risk Evaluation Model of Project Financing Based on Neural Network," in *2007 IEEE International Conference on Grey Systems and Intelligent Services*, pp. 1072–1076, 2007.

[39] A. Menon, S. Singh, and H. Parekh, "A Review of Stock Market Prediction Using Neural Networks," in *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, pp. 1–6, 2019.

[40] J. Wu and Y. He, "Prediction of GDP in Time Series Data Based on Neural Network Model," in *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pp. 20–23, 2021.

[41] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," 2008.

[42] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist, "Network on Chip : An Architecture for Billion Transistor Era," in *NorCHIP 2000*, IEEE, 2000.

[43] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A Network on Chip Architecture and Design Methodology," in *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, pp. 117–124, 2002.

[44] J. Balfour and W. J. Dally, "Design Tradeoffs For Tiled CMP On-chip Networks," 2006.

[45] Q. Yang and Z. Wu, "An Improved Mesh Topology and Its Routing Algorithm for NoC," in *2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1–4, 2010.

[46] Y. Salah, M. Atri, and R. Tourki, "Design of a 2D Mesh-Torus Router for Network on Chip," in *2007 IEEE International Symposium on Signal Processing and Information Technology*, pp. 626–631, 2007.

[47] S. Bourduas and Z. Zilic, "A Hybrid Ring/Mesh Interconnect for Network-on-Chip Using Hierarchical Rings for Global Routing," in *First International Symposium on Networks-on-Chip (NOCS'07)*, pp. 195–204, 2007.

[48] G. Xiaopeng, Z. Zhe, and L. Xiang, "Round Robin Arbiters for Virtual Channel Router," in *The Proceedings of the Multiconference on "Computational Engineering in Systems Applications"*, vol. 2, pp. 1610–1614, 2006.

[49] F. Zhizhou and L. Xiang, "The Design and Implementation of Arbiters for Network-on-chips," in *2nd International Conference on Industrial and Information Systems*, IEEE, 2010.

[50] A. S. Prakash and C. P. Ravikumar, "VLSI Implementation of a Wormhole Router Using Virtual Channels," in *Proceedings of TENCON'94 - 1994 IEEE Region 10's 9th*

Annual International Conference on: 'Frontiers of Computer Technology', pp. 1035–1039 vol.2, 1994.

[51] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application Specific Routing Algorithms for Networks on Chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 316–330, 2009.

[52] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "A Method to Remove Deadlocks in Networks-on-Chips with Wormhole Flow Control," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 1625–1628, 2010.

[53] R. V. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms For Mesh Networks," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 848–864, 1995.

[54] R. V. Boppana and S. Chalasani, "A Framework For Designing Deadlock-Free Wormhole Routing Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 2, pp. 169–183, 1996.

[55] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu, "Comparison Research between XY and Odd-Even Routing Algorithm of a 2-Dimension 3X3 Mesh Topology Network-on-Chip," in *2009 WRI Global Congress on Intelligent Systems*, vol. 3, pp. 329–333, 2009.

[56] C. Ge-Ming, "The Odd-Even Turn Model For Adaptive Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, 2000.

[57] J. Khichar, S. Choudhary, and R. Mahar, "Fault Tolerant Dynamic XY-YX Routing Algorithm For Network on-Chip Architecture," in *2017 International Conference on Intelligent Computing and Control (I2C2)*, pp. 1–6, 2017.

[58] J. An, H. You, J. Sun, and J. Cao, "Fault Tolerant XY-YX Routing Algorithm Supporting Backtracking Strategy for NoC," in *2021 IEEE Intl Conf on*

Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), pp. 632–635, 2021.

[59] Y. Kurokawa and M. Fukushi, "XY Based Fault-Tolerant Routing with the Passage of Faulty Nodes," in *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 99–104, 2018.

[60] J. D. Day and H. Zimmermann, "The OSI Reference Model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[61] J. R. Stevens, A. Ranjan, and A. Raghunathan, "AxBA: An Approximate Bus Architecture Framework," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ACM, 2018.

[62] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures," in *44th Annual International Symposium on Computer Architecture*, ACM, 2017.

[63] V. Y. Raparti and S. Pasricha, "DAPPER: Data Aware Approximate NoC for GPGPU Architectures," in *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, IEEE, 2018.

[64] Y. Chen and A. Louri, "An Approximate Communication Framework for Network-on-Chips," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1434–1446, 2020.

[65] L. Wang, X. Wang, and Y. Wang, "ABDTR: Approximation-Based Dynamic Traffic Regulation for Networks-on-Chip Systems," in *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 153–160, 2017.

[66] G. Ascia, V. Catania, S. Monteleone, M. Palesi, D. Patti, and J. Jose, "Improving Energy Consumption of NoC Based Architectures Through Approximate Communication," in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4, 2018.

[67] A. Ben Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano, "AxNoC: Low-power Approximate Network-on-Chips using Critical-Path Isolation," in *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, IEEE, 2018.

[68] L. Wang, Y. Wang, and X. Wang, "An Approximate Multiplane Network-on-Chip," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020.

[69] M. Hayenga, N. E. Jerger, and M. Lipasti, "SCARAB: A Single Cycle Adaptive Routing and Bufferless Network," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ACM Press, 2009.

[70] S. Xiao, X. Wang, M. Palesi, A. K. Singh, L. Wang, and T. Mak, "On Performance Optimization and Quality Control for Approximate-Communication-Enabled Networks-on-Chip," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1817–1830, 2021.

[71] W. Baek and T. M. Chilimbi, "Green: A Framework for Supporting Energy-Conscious Programming Using Controlled Approximation," 2010.

[72] J. Lee, C. Nicopoulos, S. J. Park, M. Swaminathan, and J. Kim, "Do We Need Wide Flits in Networks-on-Chip?," in *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2013.

[73] S. Han, J. Lee, and K. Choi, "Tree-Mesh Heterogeneous Topology for Low-Latency NoC," in *2014 International Workshop on Network on Chip Architectures*, ACM Press, 2014.

[74] J. Han, "Introduction to approximate computing," in *IEEE 34th VLSI Test Symposium (VTS)*, IEEE, 2016.

[75] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, "Chapter 2 - Link-Level Flow Control and Buffering," in *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, pp. 11–35, New York, NY: Springer New York, 2015.

[76] M. Abdelrasoul, M. Ragab, and V. Goulart, "Evaluation of the Scalability of Round Robin Arbiters for NoC Routers on FPGA," in *IEEE 7th International Symposium on Embedded Multicore SoCs*, IEEE, 2013.

[77] B. Jacob, S. W. Ng, and D. T. Wang, "Chapter 22 - The Cache Layer," in *Memory Systems* (B. Jacob, S. W. Ng, and D. T. Wang, eds.), pp. 731–745, San Francisco: Morgan Kaufmann, 2008.

[78] C. Hamacher, Z. Vranesic, S. Zaky, and N. Manjikian, "Chapter 8 - The Memory System," in *Computer Organization and Embedded Systems*, McGraw-Hill Publishing, 2011.

[79] Xilinx, "UltraScale Architecture Configurable Logic Block User Guide," 2017.

[80] J. Lewis, "Randomization Using RandomPkg," 2020.

[81] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, p. 3–30, 1998.

[82] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, 2011.

[83] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve, "gem5-approxilyzer: An open-source tool for application-level soft error analysis," *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 214–221, 2019.

[84] C. Lattner, "LLVM: An Infrastructure for Multi-Stage Optimization," Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL, Dec 2002. *See* `http://llvm.org`.

[85] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 33–42, 2009.

[86] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An Open, Extensible and Cycle-Accurate Network on Chip Simulator," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 162–163, 2015.

[87] J. Rettkowski and D. Göhringer, "RAR-NoC: A Reconfigurable and Adaptive Routable Network-on-Chip for FPGA-based Multiprocessor Systems," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pp. 1–6, 2014.

[88] G. M. Link and N. Vijaykrishnan, "Hotspot Prevention Through Runtime Reconfiguration in Network-on-Chip," in *Design, Automation and Test in Europe*, pp. 648–649 Vol. 1, 2005.

[89] S. A. M. Junos Yunus, M. N. Marsono, and I. Ibrahim, "Modeling Router Hotspots on Network-on-Chip," in *13th International Conference on Advanced Communication Technology (ICACT2011)*, pp. 896–900, 2011.