ABSTRACT

Title of thesis: AN EFFICIENT NETWORK ON CHIP (NoC) FOR A PARALLEL, LOW-POWER, LOW-AREA HOMOGENEOUS MANY-CORE DSP PLATFORM James Darin Chandler, Jr., Master of Science, 2012

Thesis directed by: Dr. Tinoosh Mohsenin Department of Computer Science and Electrical Engineering

This thesis presents an NoC architecture that is optimized for a course-grained, deterministic many core DSP platform supporting up to 256 cores. The proposed network supports both local and long-distance communication in the event that large applications or multiple smaller applications are mapped onto the platform by means of a hierarchical cluster topology.

The NoC is designed to optimize the area- and power-to-performance ratio through implementing the following key characteristics: low hop-count long distance communication, optimized flit buffer size, efficient virtual channel implementation, and a highly restricted virtual channel flow control.

The NoC architecture is implemented in 65 nm CMOS technology with a nominal supply voltage of 1V. Place and Route results show that the proposed architecture saves up to 33% in area and up to 87.6% in energy-per-flit in comparison to some currently-implemented NoCs. Through several traffic pattern tests on a network of 16 cores, the NoC attains a throughput of up to 21.7Gbps. A 256-point
FFT mapped onto 16 cores executes in $4.3\mu s$ and dissipates 0.649W. This is an improvement of 46% and 81% in latency and power dissipation over a 256-point Xilinx FFT IP Core implemented on a Virtex 6 FPGA.
AN EFFICIENT NETWORK ON CHIP TARGETED TO A PARALLEL, LOW-POWER, LOW-AREA HOMOGENOUS MANY-CORE DSP PLATFORM

by

James Darin Chandler, Jr.

Thesis submitted to the Faculty of the Graduate School of the University of Maryland, Baltimore County in partial fulfillment of the requirements for the degree of Master of Science 2012

Advisory Committee:
Dr. Tinoosh Mohsenin, Chair/Advisor
Dr. Mohamed Younis
Dr. Chintan Patel
© Copyright by
James Darin Chandler, Jr.
2012
Dedication

It goes without saying that this thesis is dedicated to the wide and talented academic community in hopes to propagate its research endeavors and its cause. However, without two people in particular, I’m not sure I would have made it this far. They are my two grandfathers, Thomas “Pappy” Chandler and Charles “Pop-pop” Vogt. They always took a deep interest in my endeavors, encouraging me to take them further. They pushed me. They provided moral and motivational support. They were there for me. They were always proud of me (I’m sure I’ll figure that out when I have children or grand-children of my own). Without dragging this on too much longer, I dedicate this thesis to Thomas Chandler and Charles Vogt. I miss you both. The journey doesn’t stop here. Thank you.
Acknowledgments

I owe thanks and gratitude to all who have made it possible for my completion of this thesis. First, I want to thank my advisor, Dr. Tinoosh Mohsenin, for her direction throughout the last year and a half. Her experience in the field and vast library of publications have most definitely helped me complete this work. Thank you for the opportunity to work on this challenging and, more importantly, interesting topic.

Dr. Younis and Dr. Patel, thank you for advising my thesis. And to the rest of the Department of Electrical and Computer Engineering, thank you. The experience here was certainly challenging, but thanks to you, very smooth. I decided to attend UMBC because of the funded Teacher’s Assistant opportunity. I’m glad I made that decision, as I’ve grown as an individual and even more as a student and researcher. Thank you all for the opportunity given to me.

I also want to thank my classmates and friends at UMBC: Dave Elrom, Kirit Chatterjee, Jordan Bisasy and Jerome . . . Vikram. There were many (probably too many) late nights and difficult stretches of work. We helped each other plow through many of the challenges of a graduate education—especially one at UMBC.

I want to also thank my professors and friends that I gained at Loyola University Maryland: Dr. Robert Pond and Professor Glenn Kohne. The preparation and support that you have given me has most certainly helped me to endure and complete a rigorous graduate education and further me along both professionally and academically. We remember you, Dr. Coyne.
To my friends that have helped, encouraged, and ensured that I finish. Steve “Ig-Fig”, Joe Doherty, Steve Zornak, Sean Gloth, Kevin Ray, Tyler Spiering, and Corey Whitby. Thank you guys for being there when I needed it most.

I owe deep thanks to my family. My mother and my father for raising me to be the person I am and providing me with all types of support to help me through this step in my life. To my sister Ashley for making sure that I never stop pushing myself. My step-father and his sons and wives–thanks for the support and understanding for the past two years. Thanks uncles, aunts, and cousins who have supported me every step of the way. To my grandmothers for the unconditional moral support and motivation. To my future wife’s parents and family–thank you for allowing me to be a part of your family. I’ve felt so welcome, comfortable, and supported. Thank you. Finally, I want to thank my grandfathers for where I am today–without them, I definitely wouldn’t be the person I am, both professionally and spiritually. Thank you all so much–I owe it all to you and love you all.

Finally, I want to thank my fiancee and soon-to-be wife, Christina Nelson. Without her, I would be lost and unmotivated. I have done all of this for our future together. Thank you for standing by me through the good and especially through the bad. Without you, this would not be possible. I love you.

To anyone I have left out, I didn’t mean it! I thank everyone that has influenced me throughout my life for this. Sincerely, from the bottom of my heart, thank you all.
# Table of Contents

List of Tables .................................................. viii
List of Figures .................................................. ix
List of Abbreviations ........................................... xi

1 Introduction .................................................. 1
   1.1 Motivation .................................................. 1
   1.2 Contributions of this Work .................................. 4
   1.3 Organization of Thesis ..................................... 5

2 Overview of NoCs and Related Work ....................... 6
   2.1 Overview .................................................. 6
       2.1.1 Topology ............................................. 6
           2.1.1.1 2-Dimensional Mesh ............................... 6
           2.1.1.2 Torus ............................................ 9
           2.1.1.3 Tree ............................................ 10
           2.1.1.4 Hybrid ......................................... 12
       2.1.2 Switching Considerations ............................... 12
           2.1.2.1 Circuit Switching ................................ 12
           2.1.2.2 Packet Switching ................................ 13
           2.1.2.3 Wormhole Switching ............................... 16
           2.1.2.4 Hybrid Switching ................................ 17
           2.1.2.5 Virtual Channels ................................ 18
       2.1.3 Synchronization ........................................ 20
       2.1.4 Power Saving Techniques ............................... 25
   2.2 Related Work ............................................... 27

3 Proposed Work ............................................... 30
   3.1 Common DSP Applications ................................... 30
       3.1.1 Original Proposed Network Architecture .................. 32
       3.1.2 Addressing Issues with the Original Architecture ....... 34
   3.2 Improved Proposed Network Architecture ..................... 35
       3.2.1 Improved VLSI Implementation Feasibility ................. 35
       3.2.2 Introduction of Wormhole Routing and Protocol ........... 36
       3.2.3 Introduction of Virtual Channels ........................ 38
       3.2.4 Restricted Routing Flow and Resultant Crossbar .......... 39
       3.2.5 Low-cost Crossbar Arbitration ........................... 41
       3.2.6 Low-cost Output Virtual Channel Arbitration ............. 41
       3.2.7 Improved Metastability Robustness ........................ 42
4 Results

4.1 VLSI Implementation .............................................. 43
4.2 Performance ......................................................... 46
  4.2.1 Test Setup .................................................... 46
  4.2.2 Random Destination with No Restrictions .................. 48
  4.2.3 Random Destination to Nearest Clockwise Router Cluster . 49
  4.2.4 Random Destination within Source Cluster ................. 49
  4.2.5 FFT Traffic Simulation ....................................... 50
  4.2.6 Other Benchmark Traffics .................................... 52
  4.2.7 FFT Mapping on DSP Platform ............................... 54
  4.2.8 Comparison to Related Work ................................. 54
4.3 Conclusion .......................................................... 57
4.4 Future Work ........................................................ 58

Bibliography ................................. 60
List of Tables

4.1 Comparison of 256-point FFT implemented on three platforms. The
die size of Xilinx FPGAs are not published. . . . . . . . . . . . . . . 57
4.2 *These numbers are normalized to 1GHz operating frequency and a
256-core network. **These numbers are from uniformly distributed
traffic at an offered load of 50%. ***Numbers taken from uniformly
distributed traffic operating at 50% load. Compared implementations
can be found in [44]. . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
List of Figures

2.1 4×4 2D Mesh. ................................................................. 7
2.2 2×4 2D Mesh. ................................................................. 7
2.3 Three examples of dimension-ordered routing employing X-first, Y-last. 8
2.4 Adaptive routing is employed to improve robustness to hardware fail-
ure [61]. ................................................................. 9
2.5 Torus topology [60]. .......................................................... 9
2.6 Torus implementing DOR [60]. .............................................. 10
2.7 A hierarchical BFT topology [49]. ......................................... 11
2.8 Hybrid topology implements buses at cluster and uses routers to con-
nect clusters [3]. ............................................................. 12
2.9 A circuit switched network sets up a path from 3 to 6 [12]. ............ 13
2.10 A 2D mesh with packet buffers at input and output ports [12]. .... 14
2.11 This is one example of how a packet can be structured. The final
composition depends on the protocol that the network implements. .... 15
2.12 The structure of a flit is similar to that of a packet. The difference
between packet and wormhole routing is that flits are immediately for-
warded to their destination, rather than waiting for the entire packet;
it work in a pipelined fashion. .............................................. 16
2.13 Router architecture implementing both circuit and packet switching
[3]. .................................................................................. 18
2.14 Packet B is blocked by packet A (a), packet A passing packet B (b),
packet A again blocked by a large packet B (c) [14]. ..................... 19
2.15 The Muller C element only changes state when both inputs are the
same. It turns on when both inputs turn on and turns off only when
both inputs are turned off [62]. .................................................. 22
2.16 This circuit, when interfaced with another, is self-timed. Extra cir-
cuitry is necessary to latch data and remove data at the appropriate
times [62]. ........................................................................ 22
2.17 Signal transitions that occur during a handshake in a clock pausing
architecture. When stretch is high, the clock has been paused on its
low phase [62]. .................................................................... 23
2.18 This is the W-Port or transmitting port. This circuit interfaces with a
remote R-Port or receiving port. It is similar to the bare handshaking
circuit seen in Figure 2.16 but with extra hardware to implement clock
pausing [62]. ................................................................. 23
2.19 This is the R-Port or receiving port. This circuit interfaces with a
W-Port or transmitting port in a remote node [62]. ......................... 24
2.20 This circuit takes as inputs the stretch signals generated by the W-
Port and R-Port. Whenever any stretch signal goes high, the clock
will then be paused on its low phase [62]. ......................... 24
2.21 A two-flop synchronizer. Research shows that any more than two flops is unnecessary. If there is a metastable event, it will resolve itself in between the two flops. .................................................. 24

2.22 EMI effects of a synchronous system (a) and an asynchronous system (b) [2]. ................................................................. 25

3.1 A 64-point FFT is mapped using 4-point FFTs per core. ............... 31

3.2 Proposed NoC platform with 64 PEs and 20 routers. Examples of possible message routes are shown. .................................................. 37

3.3 Topology of old architecture (left) vs. new architecture (right). ...... 38

3.4 High-level view of router architecture (a), block-diagram of a port (b), and block diagram of an input virtual channel (c). ................. 40

4.1 New proposed network architecture router placed-and-routed. Occupies an area of 0.073mm$^2$. ..................................................... 44

4.2 Percent power (a) and area (b) breakdown for the router. ............. 45

4.3 Average latency of the test results for the three uniform random traffic simulations. ......................................................... 48

4.4 Average throughput of the test results for the three uniform random traffic simulations. ..................................................... 49

4.5 Measured average latency for the FFT traffic pattern test. ............ 51

4.6 Measured average throughput for the FFT traffic pattern test. ........ 51

4.7 Measured average latency for benchmark traffic patterns. ............ 53

4.8 Measured average throughput for benchmark traffic patterns. ....... 54

4.9 Comparison of percent area occupation of the network to the platform (a) and energy-per-flit (nJ) of currently implemented networks (b). 56
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>GALS</td>
<td>Globally Asynchronous, Locally Synchronous</td>
</tr>
<tr>
<td>NoC</td>
<td>Network on Chip</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor/Processing</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>2D</td>
<td>2-Dimensional</td>
</tr>
<tr>
<td>DOR</td>
<td>Dimension-Ordered-Routing</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>Flit</td>
<td>Flow control digit</td>
</tr>
<tr>
<td>VC</td>
<td>Virtual Channel</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out [buffer]</td>
</tr>
<tr>
<td>BOP</td>
<td>Beginning of Packet</td>
</tr>
<tr>
<td>EOP</td>
<td>End of Packet</td>
</tr>
<tr>
<td>XBAR</td>
<td>Crossbar</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean Time Between Failure</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>MIN</td>
<td>Multi Stage Interconnect</td>
</tr>
<tr>
<td>GS</td>
<td>Guaranteed Service</td>
</tr>
<tr>
<td>BE</td>
<td>Best Effort</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Many core platforms have gained popularity in bridging the gap between highly flexible FPGAs that dissipate significant power and inflexible ASICs that are extremely power efficient. Many core platforms that integrate heterogeneous cores are known as systems-on-chip (SoCs). As SoCs become more prevalent, there is an increasing need to more efficiently integrate and connect multiple functional cores, where a single core can be anything such as a microcontroller, FPGA, memory block, or any custom hardware [16][44]. Classically, functional cores are interconnected using bus structures [48][54][4]. As contemporary SoCs grow larger, several problems arise with synchronous data buses. Most importantly, propagation medium arbitration is very difficult; the bus controller must operate at extreme speeds in order to keep pace with the bandwidth that many processing elements offer [48]. Bus width is also a constraint; some functional cores may require the transmission of very large chunks of data. Allowing bus widths to grow to hundreds of bits is impractical [20].

Electrical properties of the wires that comprise bus interconnects further emphasize these aforementioned issues as buses lengthen to accommodate large chip sizes. As CMOS processes shrink, gate delays decrease; this is not true for wires. Wires, in fact, are becoming a serious cause of concern for SoC designers [22][33].
Wire delays remain constant only when repeaters are used. In turn, wire delays scale upwards relative to gate delays. Additionally, due to the structure of buses and the number of elements connected to them, parasitic capacitance is significant [44]. The problem of high resistance and capacitance paired together results in unacceptable propagation delays [54][4]. High propagation delays on long wires with undesirable parasitic characteristics cause them to behave as lossy transmission lines [45]. As a result, bus wires generate significant electromagnetic radiation and cause unwanted crosstalk [2][15]. As a result, global clocks for SoCs that utilize large buses are limited in their operational speed [20].

The synchronicity of buses implies that an SoC is driven by a single clock. When considering the high capacitance that buses exhibit due to long wire length and heavy loading, high synchronous clock speeds dissipate significant amounts of power—up to 45% of a chip’s total power [57]. Since wire delays are such a significant factor, global clock trees are extraordinarily complex in terms of their layout. With clock speed increases through each process progression, clock skew is difficult to resolve in large global clock trees [33][2]. It is therefore impractical to continue to design SoCs with a single coherent clock [48].

An architecture that effectively combats the issue of a power hungry global clock is a Globally Asynchronous, Locally Synchronous (GALS) architecture. A GALS design splits each functional core into separate clock domains. First, this eliminates the need to create and fine tune a global clock distribution network [44]. One significant advantage of each core operating in its personal clock domain is that the core can operate at its optimal clock rate [2]. As a result, the slowest core will no
longer be in the SoCs critical path. Another advantage is that GALS architectures create the ability to shut off or scale the clock rate at each “synchronous island;” this strategy is one way in which GALS architectures limit power consumption [44][57][30][57]. Finally, when designing an SoC with a fully coherent clock, any changes made to the design necessitates a re-synthesis of the entire clock tree [2]. Thus, the implementation of a GALS SoC ameliorates the design process when a portion of a design must be changed. Additionally, the implementation of a GALS architecture effectively eliminates the ability to use a synchronous bus as the interconnect between functional cores; each core will be operating in a separate clock domain, be it in frequency, phase, or both.

Given that each functional core operates in its own clock domain in a GALS design, measures must be taken to synchronize the data transfer between each domain. Researchers have begun adapting interconnection techniques from parallel supercomputing to network-on-chip (NoC) designs [20]. Such areas of adaptation and optimization in NoCs are topology, routing algorithm, protocol, and most importantly, the router architecture that implements these parameters. Constraints of VLSI implementations limit the extent to which ideas can be adapted from parallel supercomputing. However, with novel designs, the performance of SoCs will see an increase in throughput to area and power efficiency.

Due to the current state of bus-based SoC architectures, pressure is put on researchers and industry to consider communication-centric designs such as NoCs [19]. One of the paramount performance gains associated with NoCs is the reduction in power consumption. This becomes even more significant given that computational
components will heavily dissipate power as well; they will operate at their optimal clock frequency. Therefore, power consumption is considered to be one of the most vital parameters to optimize in most designs in order to decrease the network-to-computation power consumption ratio [17].

1.2 Contributions of this Work

Most current research adapts the GALS NoC design paradigm to general-purpose, heterogenous-core SoCs. This thesis presents an NoC architecture that is optimized for a homogeneous-core, parallel, course-grained DSP platform while maintaining low-power and low-area costs. This architecture is based on an original design previously implemented to target extremely low area and power overhead. Both networks support both local and long-distance communication in the event that large applications or multiple smaller applications are mapped onto the DSP platform by means of a hierarchical cluster topology. As a result, long-distance communication exhibits a lower hop-count than that of flat network topologies, especially at larger scales. To optimize the throughput efficiency of the newly proposed network architecture, a router implements the following key characteristics: low hop-count long distance communication, optimized flit buffer size, efficient virtual channel implementation, and a highly restricted virtual channel flow control. Performance is validated through the testing of several traffic patterns with varying localities. An FFT is programmed and simulated on 16-core DSP platform integrated by means of the proposed NoC. Lastly, the proposed network architecture is
described in Verilog, synthesized to a gate netlist, and placed and routed in 65 nm
technology.

1.3 Organization of Thesis

Chapter 2 begins with an overview of NoC design choices and will end with
current work in the NoC field. Chapter 3 will discuss the original network architec-
ture as well as the newly proposed NoC architecture. Finally, Chapter 3 discusses
performance and VLSI implementation results, concludes the thesis, and illustrates
avenues of future work for the proposed NoC.
Chapter 2

Overview of NoCs and Related Work

2.1 Overview

New interconnection design strategies suggest adapting parallel supercomputing design methodologies to SoCs [20][3]. Such aspects of supercomputer interconnects are topology, routing, and protocol. First, topologies will be discussed. The overview discussion will conclude with topology-independent NoC parameters. This discussion will remain with the most popular design choices and their tradeoffs.

2.1.1 Topology

2.1.1.1 2-Dimensional Mesh

The optimal topology for an NoC is a popularly researched topic [37]. The most popular topology, due to its simplicity, is the 2-dimensional (2D) mesh. This topology can take on several “shapes.” It can be square, where the number of functional cores on each side is X, and the number of elements is $X^2$. Fig. 2.1 shows a $4 \times 4$ 2D mesh. A 2D mesh can be rectangular as well, with the number of elements on two sides equal to X and Y and the number of elements is $X \times Y$. Figure 2.2 shows an example of a $2 \times 4$ rectangular 2D mesh. Lastly, the 2D mesh can take on irregular shape, where there is no uniformity in the layout.
No matter the shape, the 2D mesh has a distinct advantage: the grid-based interconnect results in high regularity in the direction and size of the links [16][48][61][36][59][53]. The uniformity of router architecture throughout the network allows for easy routing algorithm implementation. The most simple of which is dimension-ordered routing (DOR), or minimum-path routing, which is a deterministic routing function [48][36][59][13]. In DOR, a message is routed in one direction (e.g. X) and then in the other direction (e.g. Y). Fig. 2.3 shows a 2D mesh implementing DOR. This algorithm results in very simple hardware and always guarantees in-order transmission, in-order arrival. With the grid-based architecture, router designs consist of up to five ports—North, East, South, West, and a port for core communications.

As the number of functional cores on-chip approaches the thousands, the relia-
bility of the CMOS process trends lower [61]. Therefore, it is becoming necessary to
design robust routing algorithms in hardware to combat potential hardware failure.
In [61], adaptive routing techniques are employed. First, a router is determined
whether or not it is active by a built-in self-testing routine. If hardware is found
to be faulty, the connected core, if any, is shut down and a West-First, North-Last
routing algorithm is employed. Fig. 2.4 shows an architecture that implements
adaptive routing. This gives the ability of a message to travel around a faulty piece
of hardware. Since robustness to hardware failure is implemented through adap-
tive routing algorithms, a large hardware area can result. Additionally, in-order
transmission, in-order arrival is nullified [39].

Given the simplicity of the arrangement of functional cores in the 2D mesh
topology, it is easily the most popular NoC implementation. With the regularity
of functional core placement, routing algorithms and router architectures are thus
simplified.
Figure 2.4: Adaptive routing is employed to improve robustness to hardware failure [61].

2.1.1.2 Torus

The torus is a topology similar to the 2D mesh, but has added complexity. Routers and functional cores are similarly arranged in a grid, however, each router on the edge of the network has a “long link” to the router on the opposite side in its respective column or row. Fig. 2.5 shows a $4 \times 4$ torus architecture. One advantage to this is that all routers are uniform; in 2D mesh, routers on the edge of the network have as many as two less ports than routers inside the network edge. Therefore, torus topologies have uniform routers no matter their placement in the network [60].

Figure 2.5: Torus topology [60].

Another advantage of the torus network topology is that it reduces the network
**diameter** [50]. The network diameter is the shortest path between two nodes that lay farthest apart [13]. This is a result of the long link that is inserted between routers that lay on edges opposite each other. A disadvantage of this wraparound link, however, is that it exhibits higher propagation delay and thus higher power dissipation [34].

Routing in the torus is very similar to that of the 2D mesh: DOR [50][29] can be used as well as a more complex adaptive routing algorithms [60][34][25]. Fig. 2.6 shows a torus architecture implementing DOR.

![Figure 2.6: Torus implementing DOR [60].](image)

**2.1.1.3 Tree**

The previously discussed topologies are *flat* in nature and thus possess no hierarchical structure. Hierarchical topologies are rarely direct networks; *switching nodes* in these architectures have the sole purpose of passing messages along, whereas *terminal nodes* have the ability to directly communicate with IP cores [13].

Tree topologies fall under the classification of multi-stage interconnection net-
works (MINs) and, depending on their organization, can be referred to as trees, fat
trees, butterfly fat trees, or k-ary n-flys (See Fig. 2.7) [45][13][9][10][11][35][49]. The
number functional cores in a tree-based network is normally a power of 2. The rout-
ing characteristic that tree networks exhibit is that, to communicate with a remote
node, the message must be sent “up” in hierarchy until the message reaches the
summit. The summit is the last switching node through which the message must
travel before being sent “down” in hierarchy to reach the destination IP core. Rout-
ing in this topology can be deterministic or adaptive, depending on the network
designer’s choice. The tradeoff here, as with any other NoC topology, will be added
hardware area and complexity versus robustness to hardware failure.

![Hierarchical BFT Topology](image)

**Figure 2.7:** A hierarchical BFT topology [49].

The metric that tree topologies specifically target to improve over flat topolo-
gies is throughput [9]. One contributing factor is a smaller network diameter than
that of flat topologies. More importantly, tree topologies can be arranged such that
there is less contention for router resources. However, these improvements come
with tradeoffs. Chip floorplans are rendered more complex because the topology
requires link paths that intersect on a two-dimensional surface [35]. Another dis-
advantage, in comparison with flat-topology networks, is that power consumption
increases due to the increased average length of inter-router links.

### 2.1.1.4 Hybrid

Hybrid architectures are comprised of combinations of any type of topology. Some examples are star-bus, spidergon-donut, mesh-of-trees, and any other combination of two or more topologies [33][51]. Topologies are combined in such a fashion as to exploit certain characteristics of different topologies at different levels [3]. For example, [3] mentions that meshes are inefficient for communication over short distances; since buses are sufficient as long as they are small, they should be used for clusters and meshes should be used to communicate over long distances (See Fig. 2.8) Parameters that a network designer would consider when creating an NoC topology are power, throughput, layout feasibility, and die area.

![Figure 2.8: Hybrid topology implements buses at cluster and uses routers to connect clusters [3].](image)

### 2.1.2 Switching Considerations

#### 2.1.2.1 Circuit Switching

Message passing in NoCs can be accomplished by circuit switching. In circuit switching, a connection between a transmitting node and receiving node is set up
allowing the data to propagate toward its destination[44][13][42]. The connection is only torn down after the destination node has received the message. Fig. 2.9 shows an example of a connected circuit from source to destination–no other sending node can use the same propagation medium while this message is in transit.

![Figure 2.9: A circuit switched network sets up a path from 3 to 6 [12].](image)

Data traffic that requires low latency and minimal jitter benefit from circuit switching [3]. This is because of the dedicated channel that a circuit gives between a source and destination. These strict traffic requirements fall under the term “Quality of Service” (QoS). However, there is a very high cost in establishing and arbitrating circuit connections; once a connection is made, the link cannot be used by any other sending node. Thus, hardware resource utilization is poor [8][12].

2.1.2.2 Packet Switching

Data communication between IP cores can also be accomplished by packet switching. In packet switching, messages are divided into fixed-length chunks called packets. As opposed to circuit switching, a packet is stored in a packet buffer and sent whenever a hardware resource is free, rather than establishing a complete
path from source to destination [44] Figure 2.10 shows a 2D mesh topology with input and output buffers. A packet is stored in these buffers before it is forwarded to its destination. Hence, this type of switching is also termed store-and-forward routing [42]. Packets are normally divided into a header packet that contains routing information followed by the data payload and are concluded with a trailer packet to close the connection [20]; one example of such a packet structure can be seen in Fig. 2.11. This results in narrower transmission channels which saves wire area. Another motivation for implementing packet switching methods is that it is generally fair in terms of arbitration and has greater resource utilization [42].

![Figure 2.10: A 2D mesh with packet buffers at input and output ports [12].](image)

With the advantages of packet switching come tradeoffs, however. The need for storing packets in intermediate buffers necessitates increased hardware area and greater router complexity than with circuit switching [44][58]. Along with increased area and complexity, energy expenditures raise with the addition of in-transit packet storage [3].

There are also performance tradeoffs involved with implementing a packet-
switched network. Whereas circuit switching guarantees bounds on latency and throughput, packet switching does not [39]. In fact, latency grows with the number of packet buffers between a source and destination node. Complex packet switching routers can implement packet dropping which further degrades latency and throughput [19]. Since packets do not require the reservation of an entire path for transmission, packets can contend for the same resource, be it a transmission channel or a packet buffer. This can cause further delays in packet delivery and even increase packet drops. With properly administered flow control schemes, packet dropping can be eliminated. However, this requires additional hardware; with the addition
of hardware to implement packet dropping and retransmission comes more complex hardware and thus a higher power and area cost [20][19].

2.1.2.3 Wormhole Switching

Another type of data switching is *wormhole switching*. This is similar to packet switching in that a message is divided into chunks. However, in wormhole switching, packets are further divided into fixed-length control units, or *flits* [44]. A full message is composed of a header flit, data flits, and a concluding trailer flit [32][42]. In a header flit, there are beginning-of-packet (BOP) bits that usually comprise of routing and arbitration information [44]. Fig. 2.12 shows an example flit structure. The trailer flit concludes the transmission via end-of-packet (EOP) bits that can be a checksum of the payload data [20][4].

![Figure 2.12: The structure of a flit is similar to that of a packet. The difference between packet and wormhole routing is that flits are immediately forwarded to their destination, rather than waiting for the entire packet; it work in a pipelined fashion.](image-url)
The advantage of wormhole routing is that an entire packet does not require being stored in a router before being forwarded to its destination. Instead, flits travel its path in a pipelined fashion; as soon as a flit is received, it is forwarded toward its destination [45][28]. Therefore, several routers may contain the information of a single message [58]. Since the entire packet need not be stored in a single router, buffer requirements are minimized [44][4][14]. The resultant router architecture is simpler and smaller in terms of hardware.

It is due to immediate flit forwarding that the latency of a wormhole switched network is lower than that of a packet switched network [8][20]. However, hardware channels are reserved for a particular message until its flits no longer require the channel [44]. Therefore, if a flit faces a busy channel, following flits are delayed at their current location [45][58]. As a result, wormhole switched networks are especially susceptible to deadlock. Deadlock is a situation where messages are blocked by another message for an indefinite amount of time [13]. There are several techniques to relieve a network of possible deadlock, one of which will be discussed in a following section: virtual channels [42].

2.1.2.4 Hybrid Switching

As with network topology, an NoC can be implemented using a hybrid of switching techniques. More efficient communication, in terms of power and hardware, can be achieved if certain architectural advantages are combined and exploited together [3]. The proposed architecture in [3] combines both circuit and packet
switching. Fig. 2.13 illustrates the router architecture implemented in [3]. Circuit switching offers a dedicated channel for data propagation but prevents other messages from using the medium. On the other hand, packet switching has high resource utilization but dissipates more power due to the presence of packet buffers. This architecture exploits the high throughput and low latency characteristics of circuit switching for data transmission and implements requests and circuit setup using the packet switched network. The result is fast data transfers that are arbitrated in an efficient manner using a packet switched network that, as a whole, dissipates very little power.

![Router architecture implementing both circuit and packet switching](image)

**Figure 2.13:** Router architecture implementing both circuit and packet switching [3].

### 2.1.2.5 Virtual Channels

A common analogy of virtual channels is that of commuter traffic: no virtual channels equvalates to a single-lane highway. If a car must make a left turn, and oncoming traffic is blocking that car, all traffic behind the turning car must wait until it has completed the turn [14]. Adding a virtual channel is similar to expanding...
the highway to two lanes; when a car must make a left turn and is blocked, traffic that must travel straight can do so by passing. Thus, active messages can pass blocked messages and use network bandwidth that would otherwise not be utilized [41]. Figure 2.14 shows an example of an architecture with vs. without virtual channels. It shows that although virtual channels are added, a very large message can still block the flow of following packets. Ultimately, hardware resources are more efficiently utilized through the addition of virtual channels [44].

![Diagram showing packet blocking and passing](image)

**Figure 2.14:** Packet B is blocked by packet A (a), packet A passing packet B (b), packet A again blocked by a large packet B (c) [14].

The addition of virtual channels comes at a cost, however. Each virtual channel is essentially an additional buffer and therefore router area increases with the number of virtual channels [15]. Since the additional chip area is comprised of buffers, power dissipation increases with the implementation of virtual channels [44]. With several logical networks, arbitration complexity inherently rises. This not only increases
hardware complexity and area, but the arbitration computation takes time and can increase network latency [58][42].

Latency, throughput, and jitter are often times at odds with each other when considering design tradeoffs. This is a problem when considering SoCs. With heterogeneous SoCs, comes heterogeneous data requirements. Fortunately, with the addition of virtual channels, priorities can be placed on each logical network [4]. Two examples of data classes are guaranteed service (GS) and best effort (BE) [41][42][7]. The GS traffic class has bounded latency requirements and thus needs to be delivered as soon as possible. On the other hand, BE traffic is delivered as fast as possible but has no latency bound requirements. With this added complexity, tradeoffs elsewhere are expected. More sophisticated arbitration schemes are necessary when considering multiple classes of data; this results in increased hardware complexity and area [14][42].

2.1.3 Synchronization

NoCs are inherently asynchronous when operating under the GALS design paradigm. GALS designs do not necessarily dictate different clock rates. A mesosynchronous is also classified within the GALS domain; each block can operate at the same clock frequency but with differing phase. Whether asynchronous or mesosynchronous, synchronization is necessary in order to communicate effectively across different clock domains. A major problem with asynchronous communications is the possibility of metastability when a signal crosses clock domains [2]. This is be-
cause clock edges cannot be guaranteed to be active within setup or hold times of synchronizing flip-flops.

The most common synchronization solution is a first-in-first-out (FIFO) buffer. Most asynchronous FIFOs are implemented with separate read and write clocks. In effect, the FIFO is an interface to the network; it collects data from a functional core using the remote write clock and the data is read by the network using the local read clock. Though this may seem to be a complete solution in itself, FIFOs contain read and write pointers that point to the buffer position from which data will be read or to where data will be written, respectively. Pointer synchronization within the FIFO is achieved through passing tokens. The motivation for pointer synchronization is that buffer reads and writes would otherwise be performed to the incorrect buffer position.

The advantage of using FIFOs for synchronization is that they are robust to metastability and thus are an attractive solution for data synchronization. However, since data is buffered between a functional core and the network, FIFOs are particularly susceptible to high latency. Also, buffers are costly in terms of silicon area and thus FIFO sizes are constrained.

FIFOs, as discussed before, are implemented as the interface between functional cores and the network. It is possible for routers to operate with separate clocks, so inter-router synchronization is necessary as well. One such solution is to use a FIFO inside of a router in source-synchronous designs. In source-synchronous designs, the sender sends a strobe, its clock, along with the data. This allows for the same solution as the core-to-router interface but instead serves as a
router-to-router interface.

Communication can be synchronized through the use of asynchronous handshaking circuits [57][4][2]. The simplest form of handshaking to implement in hardware is the four-phase protocol [54][43]. This protocol requires four signal transitions: REQuest low-high, ACKnowledge low-high, req high-low, and finally ack high-low. In [43] and [62], Muller C elements are used to implement the four-phase protocol. Fig. 2.15 shows the truth table for a Muller C element and Fig. 2.16 shows the self-timed handshaking interface circuit. These circuit elements allow for a self-timed handshaking operation. Therefore, it is fairly insensitive to link length and delay.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 2.15:** The Muller C element only changes state when both inputs are the same. It turns on when both inputs turn on and turns off only when both inputs are turned off [62].

**Figure 2.16:** This circuit, when interfaced with another, is self-timed. Extra circuitry is necessary to latch data and remove data at the appropriate times [62].
Asynchronous handshakes are too threatened by metastability. A common solution with network routers are pausable or stretchable clocks [30][40]. Clock stretching is implemented by delay lines and small arbiters that guarantee latching of data when the handshaking routine has completed. Fig. 2.17 shows the signal transition graph of one implemented version of clock pausing. Figs. 2.18, 2.19, and 2.20 show the circuitry responsible for implementing clock pausing. Another solution is to share tokens with neighbors to synchronize transmission slots; this is similar to the synchronous time-division multiplexing (TDM) arbitration scheme [19].

**Figure 2.17:** Signal transitions that occur during a handshake in a clock pausing architecture. When stretch is high, the clock has been paused on its low phase [62].

**Figure 2.18:** This is the W-Port or transmitting port. This circuit interfaces with a remote R-Port or receiving port. It is similar to the bare handshaking circuit seen in Figure 2.16 but with extra hardware to implement clock pausing [62].

A very robust and simple strategy to reduce the probability of metastability
Figure 2.19: This is the R-Port or receiving port. This circuit interfaces with a W-Port or transmitting port in a remote node [62].

Figure 2.20: This circuit takes as inputs the stretch signals generated by the W-Port and R-Port. Whenever any stretch signal goes high, the clock will then be paused on its low phase [62].

is to cascade synchronizing flip-flops; this allows the metastable state to resolve between subsequent flip-flops [1][40]. Figure 2.21 shows a two-flop synchronizer. Since data must propagate through multiple flip-flops, latency of the asynchronous data increases with the number of flip-flops. There are more sophisticated solutions to the metastability problem inherent in asynchronous circuits.

Figure 2.21: A two-flop synchronizer. Research shows that any more than two flops is unnecessary. If there is a metastable event, it will resolve itself in between the two flops.

Finally, the switching activity of asynchronous circuits are spread over time,
rather than localized to a single clock edge, dynamic power and EMI effects are
generally reduced or spread out, as seen in Fig. 2.22 [54][2]. However, with the
extra hardware that implements handshaking, clock pausing, FIFOs, or any other
synchronization solution, comes silicon area overheads that are not otherwise present
in synchronous designs [21][62].

Figure 2.22: EMI effects of a synchronous system (a) and an asynchronous system (b) [2].

2.1.4 Power Saving Techniques

A priority in designing NoCs is minimizing power dissipation [17]. In the VLSI
design process, power is reduced by lowering voltage swings, lowering capacitance
switching, and lowering switching activity [30]. Since NoCs operate under the GALS
paradigm, NoCs are created in effort to not only improve messaging performance,
but to significantly reduce the power consumed by core interconnects [30]. This is due to NoCs inherent system scalability and smaller interconnect delay [63]. Removing the need for a coherent global clock can significantly reduce power for a number of reasons. First, dynamic voltage and frequency scaling can be implemented to reduce the operating frequency of a local clock when computational needs are low [45]. Researchers have found that, in systems with a single global clock, the clock can dissipate up to 40% of a chip’s total power. A functional core or router in the GALS architecture can not only lower its clock rate by doing so, but also its switching activity or *activation rate*, further reducing power consumption [47]. Another strategy for saving power when designing a GALS architecture is to implement low-level voltage swings on data link [52]. However, with increasing circuitry and complex power-saving strategies come disadvantages. Low-level signaling circuits are highly susceptible to noise, since the noise margin is already very small. Since extra circuitry is designed to limit power consumption, area can possibly increase. As always when designing any architecture, there will be tradeoffs in different areas.

Aside from the inherent power savings from implementing a GALS architecture, power can be minimized at every level of NoC design: topology, switching, arbitration, and low-level circuit designs. In terms of topology, power is reduced mainly by minimizing the lengths of the links between neighboring routers. The 2D mesh and torus both have small link lengths relative to other designs [33]. However, if the 2D mesh network diameter is large, hop count increases for long distance traffic and thus contributes to higher power dissipation. Since hop-count can affect power performance, selecting the appropriate routing protocol for the application
is important; to minimize power consumption, messages should travel the shortest path available to the destination [8]. Pressure to save power here is also placed on the programmer of these SoCs to map an application correctly; programs should operate with high levels of locality or hot-spots in order to exploit the benefits created by a hardware designer [44]. It is clear that there are a multitude of tradeoffs in designing the topology and routing scheme of a network.

Power consumption can be minimized through well-designed switching protocols. Virtual channels, whereas they provide a significant increase in performance, they also increase on-chip buffer area. Therefore, with a greater number of virtual channels, on-chip power dissipation increases [44]. Another switching protocol that can significantly increase power consumption is packet dropping and recovery [15]. Therefore, when implementing a switching or routing protocol, there is a balance to strike. For example, packets could be delivered in-order with a deterministic routing pattern with virtual channels to avoid deadlock and packet dropping. On the other hand, out-of-order packet delivery can be implemented with an adaptive routing protocol which would benefit from a packet-dropping system. Each architecture has significant advantages and disadvantages; it depends on the system on which it is being implemented and the purpose for which it will be used.

### 2.2 Related Work

Current research in NoCs proposes topologies, router architectures, and VLSI techniques that result in more efficient architectures, depending on their target
The most popular topology in NoCs is the 2D mesh. [20] proposes CLICHÉ, a scalable 2D mesh in which all routers, except those at the edges, comprise of four neighbor routers and one resource. Similar to the 2D mesh, [15] proposes a Torus topology, where nodes at the edges of a network are connected through a 'long link.' [31] describes a topology that takes the shape of an octagon—each node within an octagon is, at most, two hops away from its destination. This topology can be extended into multiple dimensions to support many cores. [44] proposes a topology that is arranged as a butterfly fat-tree, where each router has two parent ports and two child ports unless it is in the first level of hierarchy where a router has four core children. [33] and [51] propose hybrid topologies, as they generally exploit characteristics of multiple topologies at different levels of hierarchy or at different regions on a chip. [5] even proposes a tool that determines the topology of an NoC that minimizes network energy expenditures.

The router architecture that implements these topologies is also a highly-researched topic. [57] and [55] propose a reconfigurable, circuit-switched NoC that can achieve a throughput of up to 1 word per cycle and very low latency transactions. Several works propose wormhole-switched architectures, which provide a compromise between circuit switching and packet switching [54][20]. The network in [56] also implements a packet-switched network, but implements it using a router that shares input queues between input ports to maximize buffer utilization and thus reduce latency and increase throughput.

As a GALS architecture requires synchronization between clock domains, sev-
eral techniques exist: bi-synchronous FIFOs, clock pausing, and simple flip-flop synchronizers. FIFOs are a very common solution, as most are implemented by-synchronously; they have a separate read and write clocks [57][55][38][46][27]. Another synchronization strategy is through self-timed circuits such as the Muller-C element. [43] and [62] use Muller-C elements to pause the local clock when a data transaction occurs; this allows data to settle at input flops to eliminate metastable events. Finally, a very robust and simple solution is to pass any foreign clock domain signal through a two-flop synchronizer [18]. Metastability is resolved between the two flip-flops and thus minimizes the possibility of latching invalid data.

A priority in designing NoCs is minimizing power dissipation [17]. In the VLSI design process, power is reduced by lowering voltage swings, lowering capacitance switching, and lowering switching activity [30]. Low voltage swings can be implemented on inter-router links, since they are generally the longest wires [52]. Topologies that exhibit long link lengths will benefit to a greater degree, however, implementing topologies with short and regular links will generally result in power savings. Switching activity can be lowered by minimizing buffer space and also by clock scaling [45][47].
Chapter 3

Proposed Work

3.1 Common DSP Applications

The proposed platform is targeted specifically to parallelizable DSP applications. Since the platform will feature up to 256 homogenous computational cores, there are common general-purpose NoC features that can be removed for this implementation. Such are commands and responses and processor interrupts—they are more useful in general-purpose platforms that integrate several different types of functional blocks. These special signals and traffics place constraints on and require more functionality from networks-on-chip, as they are a different class of traffic and generally require extremely low-latency delivery. As a result, only data traffic is delivered between cores and thus creates a path by which a very efficient NoC can be designed for the proposed DSP platform.

The FFT is one of the most widely-implemented mathematical algorithm on DSP processors. To create a design that effectively minimizes FFT computation time would likely improve the execution speed for most other DSP applications due to the complexity computation and communication patterns. Other such DSP applications that benefit from this optimization are FIR filters, IIR filters, sorts, and any other algorithm that will most-likely feature highly-localized traffic.

First, an FFT is mapped onto 16 cores. For the purpose of illustration, we
will assume that a processing element only has enough memory to support a 4-point FFT. Figure 3.1 shows a 64-point FFT mapping to 16 cores using this assumption. The ideas presented with the 64-point FFT translate directly to a system that implements cores with larger memories and therefore larger FFTs.

![Diagram of 64-point FFT mapping using 4-point FFTs per core.](image)

**Figure 3.1:** A 64-point FFT is mapped using 4-point FFTs per core.

The most logical approach for any application mapping is to localize high traffic. This reduces average latency, increases throughput, and reduces slower global traffic. Therefore, this FFT mapping initially maps local traffic together. As the FFT calculation progresses to its later stages, traffic begins to travel long distances.

As a result of the mapping in Figure 3.1, it is clear that most traffic is relatively
local. It is not until the last shuffle, or last few shuffles, that computation cores must communicate long-distance. Long-distance communication is more pronounced in larger FFTs. Therefore, the network should be built to optimize both short- and long-distance communications.

3.1.1 Original Proposed Network Architecture

A network architecture is designed that is suitable for integrating many processing elements with a low-area and low-power overhead [6]. The following design is the baseline on which the newly proposed architecture improves. To achieve the low-power and low-area, the network implements several key features. First, it implements direct core-to-core communications within a cluster of four which bypasses the router. This shows the strong emphasis on low-latency and high-throughput local traffic while dramatically reducing the size of the router; there are less ports than otherwise. Inter-cluster traffic is implemented by a hierarchical cluster topology which provides for less average hops-per-message than flat topologies such as a 2D mesh or torus. Also, buffer space is kept at a minimum in order to conserve silicon and power consumption. In addition, contention is solved through an asynchronous, first-come-first-serve queue which results in less hardware when compared to other arbitration schemes.

The proposed network topology takes inspiration from the 4-ary tree architecture [26]. The advantage of a tree network is that the worst-case message is \((\log_2 N - 1)\) hops away, where \(N\) is the number of network nodes. For a tree archi-
architecture, leaf nodes must talk up in hierarchy in order to communicate with any other node. Since most applications for our many core chip tend to map to a local region, a traffic burst with only one path to all other nodes, through a parent router, can create severe bottleneck. This architecture relieves this problem because a single core in a cluster of four can communicate to any other core within its cluster. This alleviates bottlenecking at a parent router.

The routing pattern in this network is static—it will always send messages from one node to another through the same path. The benefit of this design is that the routing is transparent to the programmer—the programmer states where the message should go and the hardware executes the request. This router implements store-and-forward routing. It stores the entire data packet and then sends it en-route to its destination after determining routing and arbitration information. Another advantage of static routing is to guarantee that each message at a destination node is received in the same order in which it was transmitted, since this platform does not implement packet dropping.

In order for a node to successfully send a message, several components must be included in the inter-node link. This includes the 16-bit data link, 8-bit source identification, 8-bit destination identification, and 4 bits of handshaking. These are all included in parallel in order improve the latency when sending a message. Serialized communication tends to introduce latency and multiple cycles of data transfer for a single message.
3.1.2 Addressing Issues with the Original Architecture

Although the original network is very small and low-power, its performance and robustness to deadlock and metastability can be significantly improved. These reasons are the primary motivation for improving the efficiency and performance in the newest network architecture.

First, the previous router implements store-and-forward routing. This means that an entire packet must be stored inside of the router before it is forwarded to its destination. This introduces significant latency, as it takes multiple clock cycles to completely store the message and begin forwarding.

The packet size in the original architecture is static. This is because the processing core, at the time, only sends 16-bit data at a time. However, since the DSP will gain the ability to dump larger chunks of data or memory as it improves, this characteristic will eventually be rendered obsolete. The packet size was initially 32 bits: 16 for data, 8 for source, and 8 for destination. A single port required these 32 bits along with 4 bits of handshaking bi-directionally. This means that, with the previous topology, a router requires 576 input-output (IO) pins. With a significant number of wires, power and area can increase.

With asynchronous handshaking that is inherent in GALS architectures comes chances of metastability. This router is susceptible to metastability; there are very little measures taken to reduce the probability of a metastable state occurring.

Finally, the original router architecture only allows the sending of one message at a time, whether or not the path from one input channel to another output
channel is blocked. This is due to the attempt to significantly reduce silicon area by implementing data transfers through one output port. Hardware resources are thus severely underutilized. The throughput and overall performance of the router can be bolstered by implementing a more efficient port and arbitration scheme.

3.2 Improved Proposed Network Architecture

This design improves upon several characteristics of the original proposed architecture. At the most basic level, the characteristics that the improved architecture aims to improve are VLSI implementation of the hierarchical topology and the hardware utilization of the router. Therefore, this architecture emphasizes throughput efficiency with respect to power and area.

3.2.1 Improved VLSI Implementation Feasibility

One manner in which the new router architecture improves over the previous is topologically by removing a port as well as removing the direct core-to-core communication. Figure 3.2 shows the hierarchical scheme of the proposed architecture as well as possible routing paths. The removal of the core-to-core communication will be explained after the virtual channel architecture is described. In terms of data ports, the router now has a maximum of seven neighbors as opposed to eight. The possible neighbors in this architecture are four children (child top-left (0), child top-right (1), child bottom-right (2), and child bottom-left (3)), two neighbor routers (clockwise router and counter-clockwise router), and one parent router. Note that
a child can either be a processing element or a router depending on the level of hierarchy in which the router resides. Specifically, when compared to the original router architecture, the “diagonal node” port has been removed. With one less port, the router will occupy less silicon area. Also, the VLSI implementation feasibility improves significantly by removing the diagonal node port. Routing the diagonal link would be impossible if trying to save area by tiling routers and computational cores in the most efficient fashion. Additionally, the propagation delay on all of the links between neighbor routers will be more uniform and thus more predictable. This is because the diagonal link would be significantly longer than the link to a clockwise or counter-clockwise neighbor.

In terms of inter-router links, the number of wires have been significantly reduced. For the new architecture, 16 pins are required for data, 2 for handshaking, and 10 for routing and flow control per port. Since there are 7 ports per router, and each set of wires is required in both directions, the number of inter-router wires for the new router are 392. This is a reduction of 184 wires per router. The links between each router, as opposed to the previous architecture, are bi-directional. The new architecture, therefore, implements full duplex communications; each port can both send and receive messages simultaneously.

3.2.2 Introduction of Wormhole Routing and Protocol

One significant improvement over the previous architecture is the implementation of wormhole routing. One flit in the new network consists of 16-bit data
and a 2-bit beginning-of-packet (BOP)/end-of-packet (EOP) field (See Fig. 2.12). When the BOP/EOP field reads 10₂, the flit is a header; it contains the source and destination of the message—each 8 bits. When the BOP/EOP field is 01₂, the flit is a trailer, and is the last chunk of 16-bit data. For cores that can dump more than 16-bit chunks of data, the BOP/EOP field will read 11₂, meaning that the flit is still in progress. The payload field of 16 bits reduces the link size and power consumption as opposed to a link of 32 bits that would transmit source, destination, and data simultaneously. It also allows for the flexibility of sending messages that comprise of data larger than 16 bits. The BOP/EOP bits, therefore, satisfy the condition that a wormhole message maintains control of a medium until the entirety of the message
Figure 3.3: Topology of old architecture (left) vs. new architecture (right).

has passed the physical resource. As a whole, this is a lighter protocol than other currently-implemented NoC architectures because there are no check-sums or other error correction schemes. This is a result of an architecture that does not implement packet-dropping—a common feature of general-purpose NoCs.

3.2.3 Introduction of Virtual Channels

To augment the wormhole routing scheme, the new router implements virtual channels. This is significant because applications on a deterministic DSP platform communicate in a bursty manner; virtual channels prevent traffic deadlock and increase hardware utilization. Specifically, there are six virtual channels at each port—input and output. Each virtual channel at an input port corresponds directly with an output port. Therefore, if a path from one input port to an output port is blocked, the input port can still receive messages and propagate them along if their paths are not blocked. Another ramification that the virtual channels have on the design is the routing calculations. Instead of calculating which router the message must travel to, the arrival input virtual channel at the next router must
be calculated. Therefore, routing calculations are made one router ahead-of-time. Figure 3.4(c) shows that the calculations are completed in a pipelined fashion. The routing calculation simply determines which virtual channel should be requested at the next router in the path to the message’s destination. This routing scheme is very low-cost, as it is a routing table implemented as a lookup table with 8 entries. There are no costly mathematical calculations that add to the router’s critical path.

The core-to-core communication link that was present in the previous network has been removed (See Fig. 3.3). This is because of the addition of virtual channels. Virtual channels exist inside of the router architecture that are exclusive to the transmission to a core within the same cluster. Therefore, the core-to-core network still exists, but in a logical manner rather than physical. The benefit to this implementation is less hardware area and less hardware complexity—there would be more hardware overhead to implement control and multiplexing between the router and the core-to-core communication.

3.2.4 Restricted Routing Flow and Resultant Crossbar

Since there are six input virtual channels and six output virtual channels, there must be a way to connect the corresponding buffers together. A crossbar solves this problem. However, traditional crossbars connect every input to every output, which is not necessary in this case since one input virtual channel corresponds with one of the other output ports. The routing calculation at the input port, therefore, determines at which virtual channel at the output port the message should arrive.
Therefore, the “pseudo-crossbar” in this router architecture only connects input virtual channels at one input port to a corresponding output port. Figure 3.4(a) shows the connection of the router’s ports to the crossbar. The fact that each input virtual channel corresponds to only one output port makes the crossbar fabric much smaller in implementation when compared to traditional crossbars. Figure 3.4(b) shows the architecture of a single port which is connected to the crossbar and a remote router or processing element.

The implication of the crossbar and 42 output virtual channels—when there are seven ports—means that a flit can ideally be output every cycle at every port.

Figure 3.4: High-level view of router architecture (a), block-diagram of a port (b), and block diagram of an input virtual channel (c).
This is a significant improvement over the old architecture, as only one message per cycle could be output per cycle for the entire router.

3.2.5 Low-cost Crossbar Arbitration

Arbitrating the crossbar are very low-cost round-robin arbiters. There are up to 42 round-robin arbiters arbitrating the crossbar since there are up to 42 output virtual channels in one router. The arbiter comprises of minimal logic since there are a large number of them. The round-robin arbiter simply uses a rotating pointer and compares it to the request-in field and if they are equal a grant is issued to that requester. Additionally, there is an internal finite-state-machine (FSM) that freezes the arbiter at the grant until a trailer flit passes through the crossbar. This satisfies the requirement that a wormhole message maintains possession of a channel until it has finished using the resource. Therefore, if the arbiter is at a slot that does not have a request, that input virtual channel misses its chance to gain access to the medium until the pointer rotates around again. Therefore, there is a tradeoff between [minimal] wasted time spots and silicon area.

3.2.6 Low-cost Output Virtual Channel Arbitration

When a flit arrives at an output port in a particular virtual channel, there are up to five other virtual channels that the message must contend with to gain access to the inter-router link. Therefore, the same round-robin arbiter that is used with the crossbar arbitration is used to arbitrate the output virtual channels. A similar
FSM is attached to the arbiter that freezes it when it grants a message. It then
the arbiter when the message passes. Therefore, as with the crossbar, a message
maintains use of the output medium—in this case, the link between routers until it
passes. The transmission medium is subsequently released from one output virtual
channel’s control, allowing the next requesting virtual channel to access the output
port link.

3.2.7 Improved Metastability Robustness

To combat metastability, any signal that crosses clock domains is passed
through a two-flop synchronizer. The chance for a circuit failure with a two-flop
synchronizer implemented in 180nm logic running at 200MHz is $10^{10^4}$ years [18]. [18]
mentions that as technologies scale down, the window for metastability shrinks, ef-
fectively decreasing the probability of a metastable event. Therefore, with the 65nm
implementation of this router, the mean time between failure (MTBF) should be
even higher.
Chapter 4

Results

4.1 VLSI Implementation

Both NoC architectures were implemented in 65 nm CMOS technology with a nominal supply voltage of 1V. We use a standard-cell RTL to GDSII flow using synthesis and automatic place and route. The hardware was developed using Verilog to describe the architecture, synthesized with Cadence RTL Compiler, and placed and routed using Cadence SOC Encounter.

The original proposed network router runs at 400MHz and occupies an area of 0.014mm$^2$. It dissipates 1.89mW, of which is 1.76mW dynamic and 0.13mW static. The original routed design routes 16 cores divided into 4 clusters. Every four clusters is connected to a router and four routers connects to a router in a similar fashion. A single core occupies 0.078 mm$^2$ and each router occupies 0.014 mm$^2$ and the entire prototype design results in a total area of 1.26 mm$^2$ [6].

The new proposed network router can run up to 1.47GHz. Figure 4.1 shows the placed-and-routed new architecture router. It occupies area of 0.0727mm$^2$. Figure 4.2 shows the area breakdown of the router that implements this network. More than half, 51.4%, of the router’s total area comes from the crossbar and its control units. Each of the 7 ports consists of between 5 and 9% of the total area.

Figure 4.2 shows the power breakdown of the router (static and dynamic). The
router dissipates a total of 8.2mW–0.599mW static and 7.59mW dynamic. Since the crossbar and crossbar control units comprise of more than half the router’s total area, it is expected that it dissipates the most power. The crossbar and crossbar control together dissipate 49.1% of the static power and 41.3% of the total dynamic power.

Figure 4.1: New proposed network architecture router placed-and-routed. Occupies an area of 0.073mm$^2$. 
Figure 4.2: Percent power (a) and area (b) breakdown for the router.

This network was combined with 16 DSP computation cores, creating a 16-node network, the total area is 1.15mm$^2$, with each router occupying 0.0727mm$^2$ and each updated DSP core occupying 0.054mm$^2$. The network occupies 25.2% of the complete platform with 0.291mm$^2$. The total power that the platform dissipates is 0.649W ($4\times P_{\text{Router}} + 16\times P_{\text{Core}}$), with the network dissipating 5.05% of the total platform power at 32.8mW.

The area and power savings of this architecture are a result of the virtual
channel routing restriction–each input virtual channel corresponds with one output port. As a result, the number of crossbar connections is drastically reduced. The pseudo-crossbar and exhibits area savings of 56% and power savings of 65% as compared to a conventional crossbar. Furthermore, the crossbar control hardware is reduced by an area of 56% as compared to controlling a full crossbar. This input virtual channel routing restriction also significantly reduces control hardware complexity at the virtual channels; since the logical interconnect is fixed, there is no need to arbitrate the virtual channels at the routers input.

4.2 Performance

4.2.1 Test Setup

In order to test injection rate, a ‘packet injector’ circuit was created. This circuit injects messages on an interval specified as a simulation parameter. This packet injector circuit is connected to the “Out Network Interface.” Each simulation was run for a total of 20,000 cycles as described in [44]. To track latency, a global counter is created and is reported when a message is injected into the network. When the entire message is received at its destination, the global counter is again reported. The difference between the final and first count is the total latency in cycles. To calculate the latency in units of time, the clock cycle period is calculated and multiplied by the number of cycles the message took to traverse the network to its destination.

To calculate throughput, the total number of bits transmitted in the test are
summed and divided by the time elapsed at the point the final message was delivered to its destination; this results in a number in units of bits/second. Latency and throughput are both plotted against offered load (as a % of capacity). The network’s capacity was calculated by determining the bandwidth of one link (port) and multiplying the link’s bandwidth by the number of ports in the network ($\text{Ports}_{\text{Router}} \times \text{#Routers}$). In each of the tests, the offered load was altered between 5 and 100% of the total capacity while measuring latency and throughput.

Several traffic patterns were tested to measure the performance of the network in different situations. First, uniform random traffic was tested. This is a worst-case scenario, as DSP applications are deterministic and thus traffic will follow the same pattern throughout the execution of a program. To maintain simplicity, a platform of 16 cores was tested. The first traffic mode in testing was random destination generation with no restriction on the destination. The second mode of traffic was random destination generation within the nearest clockwise router cluster. The third traffic test was randomly generated destinations within the source’s cluster. To gain an understanding of how this network performs in comparison to other current architectures, other common traffic benchmarks were run: $\text{bit-complement}$, $\text{bit-reverse}$, $\text{bit-transpose}$, $\text{tornado}$, and $\text{neighbor}$ as defined in [24]. Then, the network is tested realistically by testing the traffic patterns the network will experience when a 64-point FFT is mapped in the fashion shown in Figure 3.1. Finally, the network is integrated with 16 DSP cores and an FFT is executed; this test result will show the execution time of the FFT, whereas the previous tests will report the average latency and throughput performance of the network.
4.2.2 Random Destination with No Restrictions

Figures 4.3 and 4.4 show latency and throughput vs. offered load. The average latency begins at 35ns before network saturation at 50% offered load. Saturation throughput for this traffic pattern is 10.9Gbps.

Figure 4.3: Average latency of the test results for the three uniform random traffic simulations.
4.2.3 Random Destination to Nearest Clockwise Router Cluster

The next mode of testing exhibits more determinism: inter-cluster clockwise traffic. It should be noted that the address of the destination node is generated randomly, as long as the address resides within the cluster is clockwise to that of the sending node. Figures 4.3 and 4.4 show the measured average latency and throughput of this mode of traffic. Pre-saturation latency is 50ns. Saturation occurs at a load of 28% and the network throughputs 6.6Gbps.

4.2.4 Random Destination within Source Cluster

Next, intra-cluster traffic was tested. In this mode of testing, messages are sent from one core to three other cores in its own cluster. In this case, as with the
previous testing case, the address of the destination node was generated randomly, as long as the address is that of a node within the same cluster as the sending node. Figures 4.3 and 4.4 show the average latency and throughput vs. injection rate, respectively. Latency before network saturation is 22ns. The network saturates with this traffic pattern at 95% load with a throughput of 21.25Gbps.

4.2.5 FFT Traffic Simulation

Since this network is targeted to deterministic DSP applications, FFT traffic patterns were tested. Fig. 3.1 shows the traffic patterns under test. The first two rounds of traffic bursts are intra-cluster messages; a core sends to another core inside of the same cluster. Figures 4.5 and 4.6 show the average latency and throughput in the first round of traffic bursting which occurs after the second stage of FFT calculations. Pre-saturation latency is 21ns. When the network saturates at an offered load of 95%, throughput plateaus at 21.7Gbps.
Figure 4.5: Measured average latency for the FFT traffic pattern test.

Figure 4.6: Measured average throughput for the FFT traffic pattern test.

For the second round of traffic bursting, after the third stage of FFT calculation, the results are expected to be similar, as the traffic pattern is again intra-
cluster. Figures 4.5 and 4.6 show the average latency and throughput for the second round of FFT bursting for a 64-point FFT. In this traffic pattern, pre-saturation latency is 21ns. Saturation occurs at an offered load of 95% with a throughput of 21.7Gbps.

The third round of FFT traffic bursting is different than the first two; messages are sent inter-cluster after the fourth FFT stage. Figures 4.5 and 4.6 show the average latency and throughput in this mode of traffic. Latency, pre-saturation is 41ns. Saturation occurs at an offered load of 28% with a throughput of 6.7Gbps.

The fourth and final round of FFT traffic bursting occurs after the fifth stage of the FFT calculation. It is similar to that of the third round; it is also inter-cluster traffic. Figures 4.5 and 4.6 illustrate the measured average latency and throughput, respectively. Pre-saturation latency is 41ns. Saturation throughput is 6.7Gbps at a saturation point of 28% offered load.

4.2.6 Other Benchmark Traffics

Figure 4.7 shows the average latency for other common benchmark traffic patterns and figure 4.8 shows the average throughput for these patterns. These patterns include \textit{bit-complement}, \textit{neighbor}, \textit{bit-reverse}, \textit{tornado}, and \textit{transpose} as they are defined in [24]. As expected, neighbor traffic performed the highest, with a saturation throughput of 21.7Gbps at a saturation point of 95% offered load. Pre-saturation, average latency for neighbor traffic was 21ns. This result conforms with the first stages of the FFT traffic tested in the previous section. Bit-reverse
traffic saturates at an offered load of 60% with a throughput of 9.83Gbps and a pre-saturation average latency of 43ns. Bit-complement and transpose traffics behaved very similarly, exhibiting an average latency of 45ns and a saturation throughput of 6.61Gbps at an offered load of 28%. Performing the lowest was tornado traffic with an average pre-saturation latency of 55ns. Tornado traffic saturated the network at 26% load with a throughput of 6.54Gbps.

Figure 4.7: Measured average latency for benchmark traffic patterns.
4.2.7 FFT Mapping on DSP Platform

Figure 3.1 shows the application mapping for the FFT implemented on 16 DSP cores. The FFT application is written in assembly, assembled into machine code, and downloaded into each core’s instruction memory. The total execution time for the 64-point FFT was 1.09\(\mu s\). The breakdown of execution time between the computation and message trafficking are 41.23% computation and 58.72% network time. For a 256-point FFT, the execution time is 4.3\(\mu s\). The mapping for this setup is the same but each core executes a 16-point FFT rather than a 4-point FFT.

4.2.8 Comparison to Related Work

Figure 4.9 shows the area and energy comparisons with existing NoC topologies and implementations [44]. Each of the architectures in Fig. 4.9 implements 6 virtual channels and wormhole routing. These were chosen as comparisons because the
The proposed network also implements 6 virtual channels and wormhole routing. BFT, or butterfly fat tree, is a hierarchical topology with processing elements being a power of four. Each parent router to processing elements has two parents. Cliché is a network architecture arranged into a 2D mesh. The folded torus architecture is arranged into a topology similar to that of a torus. OCTO is a hybrid hierarchical topology where 8 elements lay in an octagon and remain at most three hops away from any node in the octagon. Octagons containing more than 8 processing elements are implemented as multi-dimensional octagons, where each similar element in each octagon being connected together. Spin is also a hierarchical topology similar to BFT. However, every node has four children and the parent is replicated four times at any level of the tree.

Figure 4.9(a) shows that this network consumes only 6% of the total chip area; when compared to Octo (44% total area), this network saves 33%. Figure 4.9(b) shows that the new proposed router architecture only expends 40nJ per-flit; when compared to Octo (280nJ per-flit), this network exhibits an energy savings of 85.7%. For energy comparisons, the proposed network was scaled to 130nm and an operating voltage of 1.2V; this is linearly with the feature size and quadratically with the voltage ($CV^2$). Although the previous proposed network occupies 5x less area, the cost of delivering one message in a network of 256 cores is 65.6nJ, which is 1.6× larger than the new proposed network.
As an additional comparison, the Xilinx FFT IP-Core implemented on a Virtex 6 executes a 256-point FFT in 8.02µs while dissipating 3.415W. Also, an ASIC that is specifically designed for FFT calculations executes a 256-point FFT in 0.311µs while dissipating 35mW and occupies 1mm² [23]. The ASIC executes the FFT 13.8× faster, dissipates less power by 18.5×, and consumes less area by 1.15×. However, the new proposed platform executes the FFT 1.87× faster while the FPGA dissipates 5.26× the power of the DSP platform. The previous platform implemented a 264-point FFT in 11.3µs, with the new platform performing 2.63× faster. These results show that this platform bridges the gap between fully programmable FPGAs and
<table>
<thead>
<tr>
<th></th>
<th>ASIC [23]</th>
<th>VIRTEX-6</th>
<th>Proposed Work</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (µs)</td>
<td>0.312</td>
<td>8.02</td>
<td>4.3</td>
<td>[−93, 87]</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>35</td>
<td>3415</td>
<td>649</td>
<td>[−95, 81]</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>1</td>
<td>n/a</td>
<td>1.15</td>
<td>[−15, n/a]</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of 256-point FFT implemented on three platforms. The die size of Xilinx FPGAs are not published.

inflexible ASICs in terms of performance and efficiency. Table 4.1 shows the latency, power, and area numbers for the Virtex-6, ASIC, and proposed DSP platform FFT execution.

Table 4.2 summarizes the throughput, area, latency, and efficiency of this and existing work. The works in this table are the same as those found in Fig. 4.9. Data in this are all normalized to a 256-core platform operating at 1GHz. A RoShaq router, as seen in [56], occupies an area of 0.091mm² and dissipates 58mW power and is implemented in 65nm CMOS. This router improves area and power by 20% and 86%, respectively. The virtual channel structure in the router gains a very high throughput for local traffics—this is the target, as applications will generally map to a local region on the platform. The throughput-per-area/energy numbers show that this network is very efficient at delivering messages when compared to other implementations.

4.3 Conclusion

This proposed network performs as desired when integrated into a course-grained, deterministic DSP platform. It saves up to 90% in energy-per-flit delivery and up to 33% in area in comparison to other NoC implementations. It is signif-
Table 4.2: *These numbers are normalized to 1GHz operating frequency and a 256-core network. **These numbers are from uniformly distributed traffic at an offered load of 50%. ***Numbers taken from uniformly distributed traffic operating at 50% load. Compared implementations can be found in [44].

Circuit technology significantly more efficient than an FPGA, as it performs 187% faster while dissipating 19.6% the power of the FPGA when executing a 64- and 256-point FFT. The high efficiency can be credited to the addition of virtual channels, while restricting their usage based on a flit’s path of travel; this results in a significantly smaller crossbar and thus much less power and area. Current research shows that general-purpose SoCs benefit greatly from communication-centric designs. However, through the results shown in this paper, it is evident NoCs can be adapted to deterministic, homogenous, many-core DSP platforms and improve efficiency with respect to power, area and performance.

4.4 Future Work

Several avenues of future work exist for the proposed NoC. One is self-timed handshaking routines such as those using MULLER-C elements. This could reduce the average number of cycles that a message experiences per handshaking routine. Another topic for future work is dynamic power saving, implemented either at the...
RTL level or by lower-level VLSI techniques. This could further increase the NoC’s energy efficiency. The “pseudo-crossbar” saves area in this architecture, but further area savings can be achieved. An extra level of multiplexing and arbitration can be added to dramatically reduce the number of crossbar connections. Finally, a skip-link or pass-through link can be added for clusters diagonal to each other. This would effectively add another port to a router but could decrease message latency for inter-cluster travel. These avenues of future work are all examinations of engineering tradeoffs; they can decrease router power and area, but could add latency to a message’s flight time.
Bibliography


63


[57] A.T. Tran, D.N. Truong, and B.M. Baas. A gals many-core heterogeneous
dsp platform with source-synchronous on-chip interconnection network. In
Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Sympos-
ium on, pages 214 –223, may 2009.

[58] D. Wiklund and Dake Liu. Socbus: switched network on chip for hard real

in clustered 2d-mesh networks-on-chip. In SoC Design Conference (ISOCC),

[60] Chang Wu, Yubai Li, and Song Chai. Design and simulation of a torus structure
and route algorithm for network on chip. In ASIC, 2007. ASICON ’07. 7th

[61] Zhen Zhang, A. Greiner, and S. Taktak. A reconfigurable routing algorithm for
a fault-tolerant 2d-mesh network-on-chip. In Design Automation Conference,

An asynchronous wrapper with novel handshake circuits for gals systems. In
Communications, Circuits and Systems and West Sino Expositions, IEEE 2002
International Conference on, volume 2, pages 1521 – 1525 vol.2, june-1 july
2002.

[63] Li Zi-di and Jiang Lin. A congestion avoidance communication protocol for
network on chip. In Intelligent Computing and Intelligent Systems (ICIS),