Exploring FPGA Overlay Network On Chip

by

CHETHAN KUMAR H B
G1502140K

A Thesis presented for the partial fullfillment of degree of
Master of Science

School of Computer Engineering
Nanyang Technological University
Singapore
MAY 2016

Abstract

Usage of On-chip networks to communicate between processors is a great way to improve performance in a Multiprocessor system. We implement a multiprocessor system which uses a Network on Chip (NoC) for establishing communication between processors (PE's) with the help of send and receive instructions. We use MIPSfpga and Flexgrip GPU soft processors as they allow implementation of customized Instructions. FlexGrip (Flexible GRaphIcs Processor for generalpurpose computing) Message Passing GPU is an efficient implementation using NOC which is a technique to overcome the shortcomings of using a shared memory. The idea behind this implementation is to provide reliable and secure communication between streaming multiprocessors. Flexgrip architecture supports addition of new instructions for inter streaming multiprocessors communication where each streaming multiprocessor has a dedicated register file. In this project, we add a new opcode to introduce send/recv functionality for streaming multiprocessor communication.

Acknowledgments

I would like to thank all the people who contributed to the work performed during this project. First and foremost, I would like to thank my dissertation advisor, **Professor Nachiket Kapre**, for accepting me into his group and providing me with his excellent and insightful guidance. During my tenure, He contributed to a rewarding graduate school experience by giving me intellectual freedom in my work. I particularly wish to thank him for arranging weekly meeting which were helpful in discussing the project to find the right path to proceed. I am truly grateful for his dedication and direction.

I would like to give a hearty thanks to my friends and parents, who supported me during my time at NTU.

Thank you everyone, it would not have been the same without support of each and every one of them.

Contents

Acknowledgments . . . 3

1 Introduction **9**

2 Background **11**
2.1 RTL coding . . . 11
2.2 Simulation . . . 11
2.3 GPU . . . 11
2.4 Message Passing Model . . . 13
2.5 MIPSfpga . . . 14
2.6 Torus NOC . . . 15

3 Flexgrip-NoC **17**
3.1 Design . . . 18
3.1.1 Message Passing Interface . . . 18
3.1.2 SEND Instruction Flow . . . 19
3.1.3 RECEIVE Instruction . . . 19
3.1.4 Proposed Designs . . . 20
3.2 Resource Utilization Of Original Flexgrip Core and SIMD NOC . . . 21
3.3 Conclusion . . . 21

4 MIPSfpga **23**
4.1 Design . . . 23
4.1.1 User Defined Instruction . . . 23
4.1.2 Send . . . 24
4.1.3 Receive . . . 25
4.2 Performance Improvements . . . 25
4.3 Methodology . . . 25
4.4 Results . . . 26
4.4.1 Sustained Rate Test . . . 27
4.4.2 Throughput Measurement . . . 27
4.4.3 Sensitivity Test . . . 28
4.4.4 Area Utilization . . . 28
4.4.5 Power measurements . . . 28

4.5 Conclusion 29

5 Optimized MIPSfpga Soft Processor **31**
5.1 Original Architecture 31
5.1.1 Execution Data Path 31
5.1.2 Memory Management Unit (MMU) 32
5.1.3 Instruction Cache and Data Cache Controllers 32
5.1.4 User Defined Instructions 32
5.1.5 Bus Interface Unit (BIU) 33
5.1.6 Register File 33
5.1.7 Multiplication and Division Unit (MDU) 33
5.1.8 Master Pipeline Control (MPC) 33
5.2 Optimized Architecture 33
5.3 Synthesis 34

6 Hierarchical Torus Networks on Chip **35**
6.1 Idea 35
6.2 Multi Layer Network Design 36
6.2.1 Node Router 36
6.2.2 Client Link 37
6.2.3 Multi-Channel Global Network 38
6.3 Deadlock Free Network 38
6.3.1 Deadlock Condition 38
6.3.2 Two Channel Local Network 38
6.4 Methodology 39
6.4.1 Simulation 40
6.4.2 Physical Layout 40
6.5 Results 41
6.5.1 Throughput Tests 42
6.5.2 Latency Analysis 44
6.5.3 Deflection Count Analysis 44
6.5.4 Physical Implementation 46

List of Figures

2.1 GPGPU Architecture 12
2.2 High-Level diagram of Hoplite router microarchitecture. Mux mapped to fractured Xilinx 5-LUTs. Processing Element (PE) implements user logic that injects and receives network traffic. 15
2.3 Interface between hoplite router and the processing element connected to it. 16

3.1 Block diagram depicting the details of the FlexGrip Streaming Multiprocessor. 17
3.2 Message-Passing Flexgrip GPU Organization. 18
3.3 Hardware Control Flow 19
3.4 Proposed design techniques (a) Single Lane (b) Multi lane without arbiter (c) Multi lane with arbiter. 20
3.5 Flexgrip Supported CUDA instructions. 22

4.1 Send Instruction Format 23
4.2 Receive Instruction Format 24
4.3 Send instruction flow 24
4.4 Multiplication Simulation 25
4.5 Sustained Rate v/s Injection Rate 27
4.6 Throughput v/s Number of cores 28
4.7 Execution rate v/s FIFO depth 29

5.1 Original MIPS Architecture 32
5.2 Optimized MIPS core 34

6.1 Comparing Flat 2D Torus with Multi-Layer NoCs. Overall system size $N \times N$, while lower-level is $d \times d$. 36
6.2 High-Level diagram of Hoplite router microarchitecture. Mux mapped to fractured Xilinx 5-LUTs. Processing Element (PE) implements user logic that injects and receives network traffic. 37
6.3 Client Link Interface 37
6.4 Deadlock condition 39
6.5 Deadlock Free Interface design 39
6.6 Two channel local network 40

6.7 Conventional Static Traffic Patterns [10] 41
6.8 Worst case latency vs. offered throughput for 16x16 system with variable local network size and local traffic pattern 42
6.9 Sustained vs. Offered Throughput in the NoCs for random traffic pattern (each point is an offered throughput) 43
6.10 Sustained vs. Offered Throughput in the NoCs across various traffic patterns (16X16 with variable local network size, each point is an offered throughput) 43
6.11 Sustained vs. Offered Throughput in the NoCs for local traffic pattern (each point is an offered throughput) 44
6.12 Density Histogram of Packet Latency for 16x16 system (variable local network size) NoC routing local traffic, offered throughput=0.1 45
6.13 Density Histogram of Packet Latency for 16x16 system NoC routing local traffic for various offered throughput 45
6.14 Percentage deflection at local and global network vs. offered throughput for local traffic pattern 46
6.15 Total deflection vs. Injection rate for 16x16 system with variable local network sz and local traffic pattern 46
6.16 Total deflection vs. offered throughput for 16x16 system with variable local network size and random traffic pattern 47
6.17 16×16 (4×4 lower level) 320 MHz chip-spanning 32b NoC on the Xilinx XC7V2000T (multi-die FPGA). 47

List of Tables

3.1 SEND instruction Format (Direct Addressing). 19
3.2 Resource Utilization on ML605 (Virtex-6) 21

4.1 Resource Utilization 29
4.2 Approximate Power Dissipation 30

5.1 Resource Utilization 34

6.1 Design-space Exploration of NoC parameters. 40

Chapter 1

Introduction

Interconnect scalability, performance, and energy efficiency are prime concerns in the design of future CMPs (chip multiprocessors). As CMPs are built with greater numbers of cores, centralized interconnects (such as crossbars or shared buses) are no longer scalable. The Network-on-Chip (NoC) is the most commonly-proposed solution [1]. It is a communication centric interconnection approach which provides a scalable infrastructure to interconnect different IPs (intellectual property) and sub-systems in a SoC (System On Chip) [5]. Moreover, NoCs can make SoCs more structured, reusable, and can also improve their performance [5]. In NoC, cores exchange packets over a network consisting of network switches and links arranged in some topology.

The demand for inter-processor communication over the past few years has been escalated which in turn has given impetus to parallelization of processes. The most common way of implemeting inter-processor communication is through shared memory.Shared memory architecture (SMA) refers to a multiprocessing design where several processors access globally shared memory. Shared memory is simple to implement because it directly accesses the address space to read from or write to a memory address but on the flipside shared memory has a bandwidth limitation as the data bus is shared among the processors. Another method to achieve energy and performance (Bandwidth) efficient inter processor communication could be message passing using NOC interface. Multi processor architecture with message passing has piece of local memory allocated for each individual processor which can be directly written by another processor through NOC interface. Message passing method is still asynchronous where receiving processor is not intimated about the arrival of data from some other processor at it's local memory. Data synchronization has to be taken care at the software/compiler level.

This work focuses on the implementation of multiprocessor systems and establishing communication between them through NOC. We use two soft processors to build multiprocessor system. 1.Flexgrip Flexgrip (FLEXible GRaphIcs Processor for general-purpose computing)[2] is a soft GPU implementation of NvidiasG80 architecture for

FPGAs. The architecture has been implemented in VHDL for a variety of parameters and can be evaluated in hardware using ML605 Virtex-6 FPGA platform.The amount of parallelism is customizable at multiple levels including the number of streaming multiprocessors(SM), scalar processors(SP) and threads. Hence, flexgrip is customizable according to specific project requirement. 2.MIPSfpga soft processor from Imagination technologies implemented in verilog which allows implementation of User Defined Instructions (UDI) through its corExtend interface. The key contributions of this report include:

(i) Implementation of new instruction set Send/Recv instructions:- We define two new customized instructions SEND/RECEIVE and add them to the existing processor instruction set. We also implement decoding logic for these new instructions in the existing instruction decoder.

(ii) Instantiations of upto 256 MIPSfpga cores and establishing communication between them through NOC :- We build a parameterized multi processor system where number of processors to be instatiated is passed as a user input and estblish communication between them through network on chip .

(iii) Compilation of MIPSfpga design on DE2-115 and Flexgrip design on ML605 boards: We compile our multi processor system RTL (Register Transfer Logic) design by targeting DE2-115 and ML605 boards to record and analyze the power and resource utilization..

Chapter 2

Background

2.1 RTL coding

The digital designers use Hardware Description Languages(HDL) to capture the behavior and structure of sytems and circuit designs. The FPGA vendors take this RTL as input and synthesize the necessary logic circuits. We implement our design in verilog and VHDL.

2.2 Simulation

Simulation is the process of creating models that mimic the behavior of the device you are designing (simulation models) and creating models to exercise the device (test benches). The simulation model need not reflect any understanding of the underlying technology, and the simulator need not know that the design is intended for any specific technology. We used modelsim and Xilinx ISIM to simulate MIPSfpga and Flexgrip designs respectively.

2.3 GPU

Initially introduced as special-purpose accelerators for graphics applications, graphics processing units (GPUs) have now emerged as general purpose computing platforms for a wide range of applications. Nowadays, GPUs are accelerating applications in platforms ranging from cars, to mobile phones and tablets, to drones and robots. Basically, a GPU is a co-processor used by a CPU to offload compute-intensive operations in order to improve the efficiency of complete system. A CPU consists of a few cores optimized for sequential serial processing, whereas GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. An overview of GPU is shown in Figure 2.1.

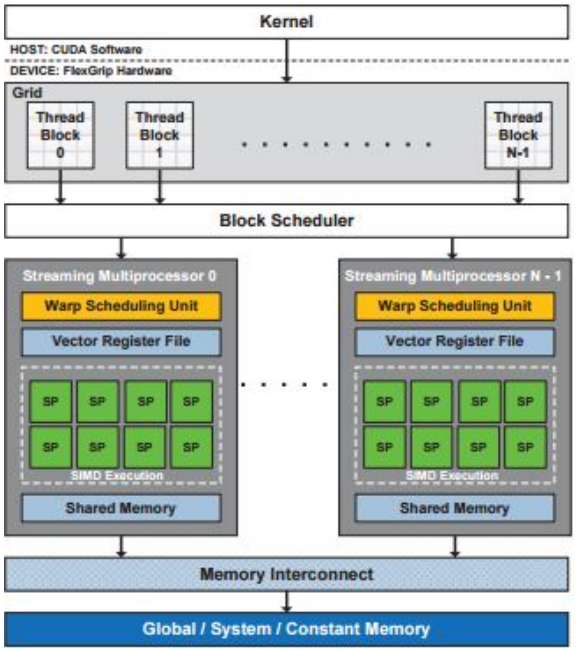

Figure 2.1: GPGPU Architecture

In terms of processor architecture, GPUs are not only not compatible with the predominant x86 instruction set, they are unlike any general purpose processors of any architecture. GPUs have hundreds of small mini-cores with very proprietary instruction sets that may change considerably with every hardware generation, as they are not exposed to the outside world, but buffered through drivers and programming interfaces. Complex, and usually never fully disclosed cache and memory architectures, coupled with unpredictable future internals of each vendors GPU family, do nothing to create a base of wellversed low-level coders. Secondly, being in a separate memory space, with totally different and disparate instruction sets, means programming the GPU heterogeneously via OpenCL or CUDA. As GPU and CPU do not share any memory, hence offloading, copying and copying back the data from GPU to CPU causes latency. Peripheral Component Interconnect express (PCIe) a high speed serial computer expansion bus, might have quadrupled its net bandwidth over the past decade, but the latency didnt improve much, since the heavy PCIe protocol takes time to process. Even if we have six or eight gigabytes of RAM on the GPU, the moment the problem gets bigger than that RAM, the performance goes down an order of magnitude, as transferring anything over PCIe lowers the speeds twentyfold compared to the on board memory.

GPGPUs have a multi-core device architecture which poses substantial parallel processing capabilities. A GPGPU is typically a GPU that performs computation of applications traditionally handled by CPU. A GPGPU consists of an array of multiprocessors which enables the device to executing multiple threads concurrently, thereby increasing

the parallelism considerably. A GPGPU is primarily comprised of an array of streaming multiprocessors (SM) with each multiprocessor consisting of multiple scalar processors (SP). Each SM processes an instruction in SIMD style, which in turn means that all the SPs in a SM processes the same instruction. Every scalar processor owns a pool of register to store operands and intermediate results which ensures protection from any data dependent hazards. Global memory facilitates inter-SM communication. A shared memory serves as a communication medium between different cores residing in the same SM. In addition, there is a read-only constant memory accessible by all the threads. The constant memory space is a cache for each SM, thus allowing fast data access as long as all threads read the same memory address.

In the CUDA programming model, the host program launches a series of kernels organized as a grid of thread blocks. A thread block represents a collection of operations which can be performed in parallel. The NVIDIA device architecture partitions thread blocks and groups them into warps, where a warp is a smaller set of simultaneous operations, some of which may be performed conditionally. Multiple warps may be assigned to a single SM and scheduled over time. To manage fine-grained scheduling, each SM is architected as a single instruction, multiple-thread (SIMT) processor. A single instruction is mapped to the scalar processors in the SM and each processor thread maintains its own program counter (PC). Every thread performs the same operation on a different set of data, but is free to independently execute data-dependent branches. Branching threads diverge from the normal execution flow and scalar processors which do not execute the branch must be marked (deactivated) during this execution. The instructions pointed to by the branching threads are executed serially, while the non-branching threads are masked.

2.4 Message Passing Model

Message passing is a technique to implement inter-processor communication. It provides a reliable way for concurrent processes to communicate. A message passing system provides primitives for sending and receiving messages. These primitives may by either synchronous or asynchronous or both. A synchronous send will not complete (will not allow the sender to proceed) until the receiving process has received the message. This allows the sender to know whether the message was received successfully or not . An asynchronous send simply queues the message for transmission without waiting for it to be received . A synchronous receive primitive will wait until there is a message to read whereas an asynchronous receive will return immediately, either with a message or to say that no message has arrived.

In message passing model, we ensure that the processors are connected through reliable communication channels, which do not lose, create or alter messages. When using this

model, processors communicate using send/receive primitives which encapsulate TCP-like communication protocols provided in modern networks. The Shared memory model abstracts a hardware shared memory made of registers. Here, the processors exchange the information using read and write operations exported by the registers. One fundamental feature of shared memory is that all communication is implicit, via loads and stores to a global address space. Here, the underlying system is responsible for deciding whether or not data is cached and for locating data in the event that it is not cached which can be expensive in terms of response time. A second important feature of shared memory is that synchronization is distinct from communication, and hence special synchronization mechanisms must be employed in addition to load and store operations in order to detect when data has been produced or consumed. Shared memory implementation needs mutexes and semaphores at the software level to effectively handle communication between SMs. In the current implementation, only independent tasks can be assigned to each streaming multiprocessor because of the lack of an efficient and reliable way of inter processor communication.

2.5 MIPSfpga

MIPSfpga is an Imagination MIPS32 microAptiv microprocessor with cache and memory management unit for educational use. It comes with complete Verilog code suitable for simulation and for implementation on a field-programmable gate array (FPGA) board. The MIPSfpga core is a version of the microAptiv UP. microAptiv processors are found in a wide variety of commercial applications including industrial, office automation, automotive, consumer electronics, and wireless communications. The MIPSfpga core is defined in the Verilog hardware description language (HDL). It is called a soft core processor because it is described in software (Verilog) instead of being fabricated on a computer chip. Figure 5.1 shows a diagram of the MIPSfpga processor.
MIPSfpga comprises approximately 12k Verilog statements and has the following features:

(i) microAptiv UP core running MIPS32 ISA with 5-stage pipeline delivering 1.5 Dhrystone MIPS/MHz

(ii) 4KB 2-way set associative instruction and data caches.

(iii) Memory management unit with 16-entry TLB.

(iv) AHB-Lite bus interface.

(v) CorExtend for user-defined instructions.

(vi) No digital signal processing extensions, Coprocessor 2 interface, or shadow registers.

2.6 Torus NOC

Torus is a topology, which is similar to the 2D-array in which nodes form a regular cyclic 2-dimensional grid. Here all routers have four connections since a torus basically is a mesh with wrap-around on the edges. A unidirectional buffered torus switch only accepts packets from two directions (and the Processing Element). Hoplite [9] is built on concept of torus. It demonstrates a remarkably lightweight design that emphasizes FPGA implementation economy by using buffer less, deflection routed approach with a 2D torus resulting in simpler cheaper switches. It removes expensive FPGA buffers, logic overheads for supporting virtual channels and complex flow control approaches and tilts the balance in favor of using as few LUTs as possible. Hoplite is easily the best matched NoC router design for FPGA implementation due to its low cost, and high performance. We show a high-level diagram of the Hoplite [9] router microarchitecture in figure 6.2.

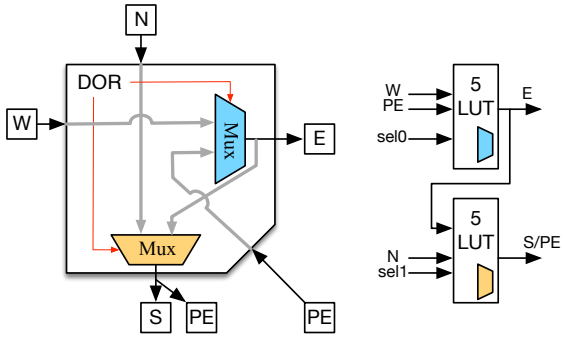

Figure 2.2: High-Level diagram of Hoplite router microarchitecture. Mux mapped to fractured Xilinx 5-LUTs. Processing Element (PE) implements user logic that injects and receives network traffic.

As we can see, it is a set of two multiplexers implementing the switch crossbar and some minimal logic for dimension ordered deflection routing. There are no FIFO buffers, or virtual channel logic thereby keeping the design simple. There are three key design principles that enable an FPGA-efficient high-speed implementation of this microarchitecture:

- Topology: Hoplite uses the unidirectional 2D torus topology thereby simplifying the switch crossbar. This makes it possible to pack each bit of the switch crossbar into two cascaded 5-LUTs (obtained by fracturing the Xilinx 6-LUT).
- Buffer less, Deflection-Routing: FIFO implementations of FPGAs make expensive use of LUT resources and are a dominant source of area requirements in [7], [6]. Hoplite uses buffer less, deflection-routing to avoid this cost entirely. This also simplifies flow control as the underlying routing algorithm simply makes decisions based on arriving packet destinations. The Hoplite design even when mapped to large FPGAs with 100s of routers, still routes at 300500 MHz.

- FPGA-aware Mapping: Hoplite exploits the fracturable 6-LUT architecture of the Xilinx FPGA, and abundance of wiring to economically map the router to FPGA logic. In Figure 6.2, the cascaded LUT connections are designed to fit both multiplexers into a single 6-LUT. To further save resources, Hoplite resource shares the South and Processing Element (PE) output ports with the same register with separate valid bits indicating the mode of use. Under this arrangement, (1) West→ South turn will also block a PE→East packet, and (2) either West→PE or North→PE exits are allowed. These routability restrictions, have a minimal 2-3% reduction in sustained rates [9] but permit an economical 1 6-LUT/bit mapping.

When evaluated against statistical traffic patterns, Hoplite demonstrates a 1.5-2× advantage over conventional buffered NoCs mapped to identical FPGAs. However, this comes at the expense of longer worst case routing latencies compared to buffered routers.

Buffer less torus has simple hand shaking mechanism between node router and the processing element as shown in figure to achieve synchronous communication. Processing elements injects next packet into the network only after receiving acknowledgment for previous packet from the router. On the other side router sends out packets to the processing elements only upon receiving destination ready signal, otherwise it will deflect the packet. 2.3. Here each node router generates acknowledgment (ack signal) to

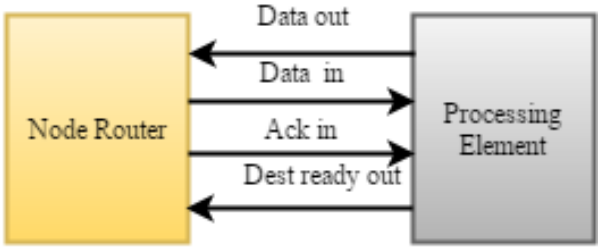

Figure 2.3: Interface between hoplite router and the processing element connected to it.

Chapter 3

Flexgrip-NoC

Flexgrip (FLEXible GRaphIcs Processor for general-purpose computing)[1] is a soft GPU implementation of NvidiasG80 architecture for FPGAs. The architecture has been implemented in VHDL for a variety of parameters and can be evaluated in hardware using ML605 Virtex-6 FPGA platform.The amount of parallelism is customizable at multiple levels including the number of streaming multiprocessors(SM), scalar processors(SP) and threads. Hence, flexgrip is customizable according to specific project requirement. With multiple SMs, there arises requirement for process synchronization which is currently resolved through shared memory model. However,it is not an efficient and a reliable method. So, we introduce an efficient alternative of using SIMD-capable NoC that can efficiently route data within the GPU using message-passing instead of synchronizing via global memory . We modify the Flexgrip SM (symmetric multi-processor) RTL to support special SEND, RECEIVE instructions to add message-passing capability to the individual SPs (single processor).

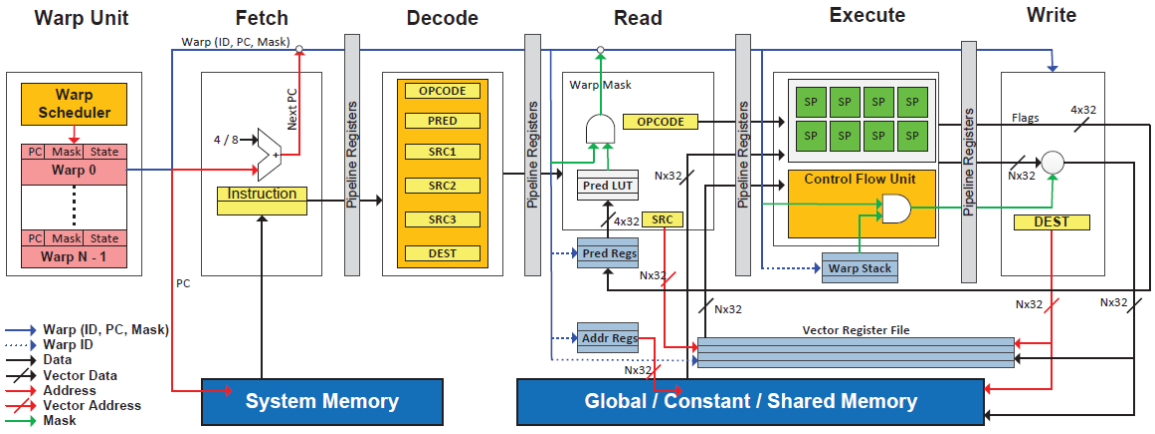

Figure 3.1: Block diagram depicting the details of the FlexGrip Streaming Multiprocessor.

3.1 Design

Message passing is a technique to implement inter-processor communication. It provides a reliable way for concurrent processes to communicate and synchronize. This message passing technique is implemented by introducing two new instructions SEND and RECV to the G80 ISA. These instructions enables inter SM communication based on SIMD-friendly Network on chip routing. Figure 3.2 shows the Message-Passing architecture for Flexgrip GPU.

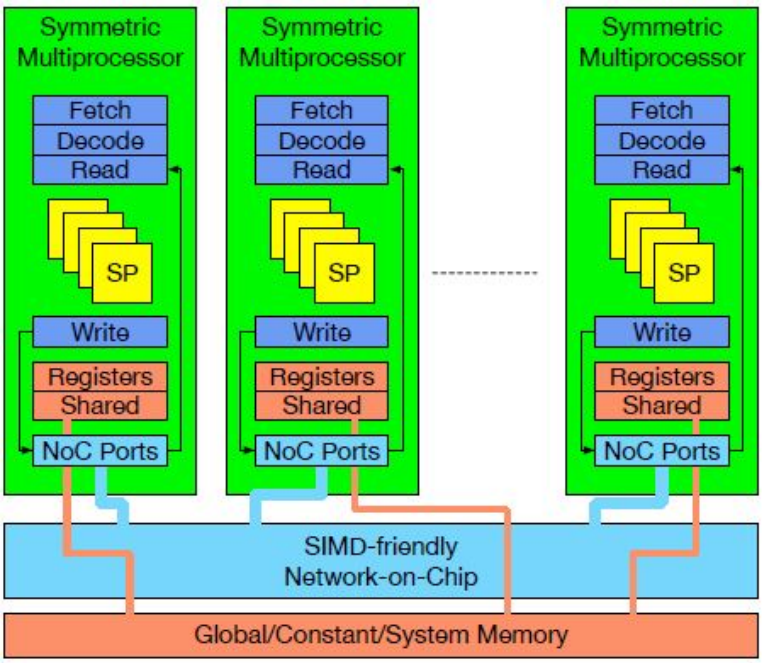

Figure 3.2: Message-Passing Flexgrip GPU Organization.

3.1.1 Message Passing Interface

We can improve performance and energy efficiency of inter-kernel communication between GPU SMs (Symmetric Multi-processors) through a SIMD-capable NoC that can efficiently route data within the GPU using message-passing instead of synchronizing via global memory. We modify the Flexgrip SM (symmetric multi-processor) RTL to support special SEND, RECEIVE instructions to add message-passing capability to the individual SPs (single processor). This architecture is shown in Figure 3.3. In this implementation, we adopt NOC based message passing interface. For reliable communication, each Streaming Multiprocessor is alloted a dedicated NOC interface system. Each NOC interface system is monitored and managed by a central NoC System, which ensures a hassle-free communication. NOC interface system is triggereed by pipeline decoder when it decodes send/receive instructions. It is responsible for encoding and decoding of NOC

packets, it has an FIFO to buffer the packets when the network is occupied. SEND command is issued when data needs to be transferred from one SM to another.

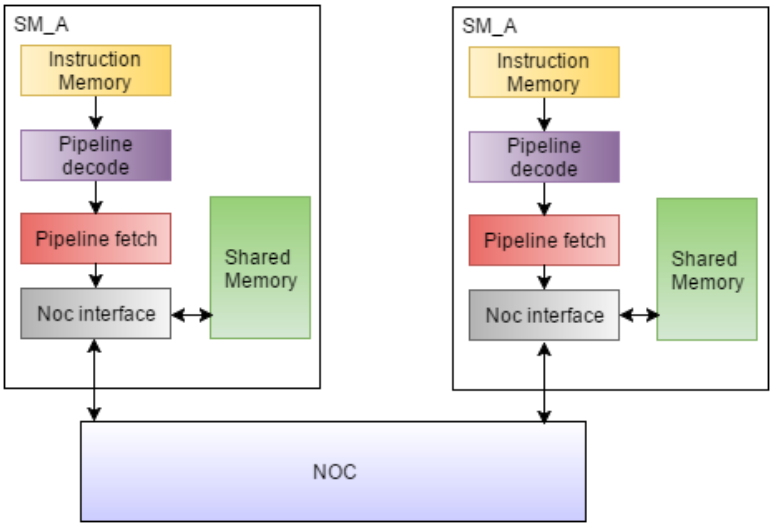

Figure 3.3: Hardware Control Flow

3.1.2 SEND Instruction Flow

When send instruction is decoded at the pipeline decoder, it intimates NOC interface system to send out the data from the source SM to the destination SM over the SIMD NOC. NOC interface system receives instruction valid, source and destination address from the pipeline decoder. Destination address contains both SM and it's shared memory address. NOC interface generates a stall signal to the pipeline decoder if there is no space available in the fifo. SEND instruction basically reads data from the source shared memory whose address is part of the instruction and writes into the destination SM shared memory through SIMD NOC. SEND instruction format is shown in Table 3.1.

3.1.3 RECEIVE Instruction

Receiver instruction is not a new instruction, we make use of existing load instruction to load data received from another SM to the register file.

Table 3.1: SEND instruction Format (Direct Addressing).

Bit Field	Description
instr in[31:28]	1100 represents SEND
source operand type	shared memory
instr in[27:18]	source addr
dest operand type	shared memory

instr in[17:2]	dest addr

3.1.4 Proposed Designs

In this work, we propose three different designs Flexgrip message passing. Designs are categorized based on the number of SIMD NOC lanes and SM shared memory slots (ports).

Single Lane

In this design, irrespective of the number of cores available in the SM, we support only one message transfer at a time between the SM's through single lane SIMD NOC. Shared memory has two ports, one port is dedicated for NOC interface and the remaining one is arbitrated among all the cores (ALU's). This design is simple to implement, however it lacks in data throughput because of the memory arbitration.

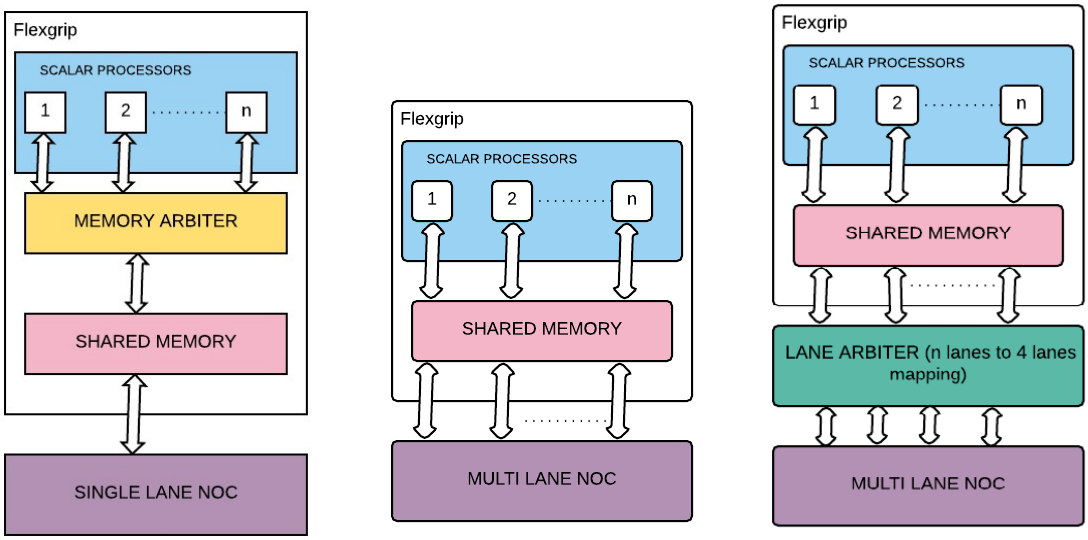

Figure 3.4: Proposed design techniques (a) Single Lane (b) Multi lane without arbiter (c) Multi lane with arbiter.

Multi Lane without arbiter

In this design, Shared memory is designed in such a way that, each SM core has it's own dedicated shared memory port. SIMD NOC support number of lanes which is equal to the number of SM cores. Each core can send out the data simultaneously through SIMD NOC. This design is efficient in terms of throughput.

Multi Lane with arbiter

This design is equivalent to the Multi lane without arbiter design except the number of SIMD lanes can be less than number of SM cores. SIMD lanes are not fixed they are parameterizable. Challenging task in this design is to map number of SM cores to the available SIMD NOC lanes.

3.2 Resource Utilization Of Original Flexgrip Core and SIMD NOC

We compile Flexgrip and SIMD NOC for ML605 using ISE 14.7 to report resource utilization. Table 5.1 depicts resource utilization of SIMD NOC and Flexgrip in terms of LUT's and FF's. From the table 5.1 it is evident that, SIMD NOC resource utilization is negligible as compare to Flexgrip. With this resource utilization data we can conclude that interfacing SIMD NOC with Flexgrip doesn't really contribute to increase in area rather it improves data throughput.

Table 3.2: Resource Utilization on ML605 (Virtex-6) .

Flexgrip					SIMD NOC				
CORES	LUTs	FFs	LUTs (%)	FFs (%)	LANES	LUTs	FFs	LUTs (%)	FFs (%)
8	71323	103776	42	33	8	511	400	<1	<1
16	113504	149297	64	46	16	1023	800	1	1

3.3 Conclusion

Adding a SIMD capable NOC based message passing system that can efficiently route data within the GPU using message-passing instead of synchronizing via global memory improves performance and energy efficiency of inter-kernel communication between GPU SMs (Symmetric Multi-processors).

Opcode	Description
I2I	Copy integer value to integer with conversion
IMUL/ IMUL32/ IMUL32I	Integer multiply
SHL	Shift left
IADD	Integer addition between two registers
GLD	Load from global memory
R2A	Move register to address register
R2G	Store to shared memory
BAR	Barrier synchronization
SHR	Shift right
BRA	Conditional branch
ISET	Integer conditional set
MOV/ MOV32	Move register to register
RET	Conditional return form kernel
MOV R, S[]	Load from shared memory
IADD, S[], R	Integer addition between shared memory and register
GST	Store to global memory
AND C[], R	Logical AND
IMAD/ IMAD32	Integer multiply-add; all register operands
SSY	Set synchronization point; used before potentially divergent instructions
IADDI	Integer addition with an immediate operand
NOP	No operation
@P	Predicated execution
MVI	Move immediate to destination
XOR	Logical XOR
IMADI/ MAD32I	Integer multiply-add with an immediate operand
LLD	Load from local memory
LST	Store to local memory
A2R	Move address register to data register

Figure 3.5: Flexgrip Supported CUDA instructions.

Chapter 4

MIPSfpga

MIPSfpga soft processor from Imagination technologies allows implementation of any new functionality via it's UDI (User Defined Instruction) interface. We implement send and receive UDI instructions to transfer data between the processors in a Multiprocessor system. MIPSfpga Multiprocessor system uses NOC (Network On Chip) as it's inter processor communication bus to enable direct data transfer between the processors. We implement the design on an Altera Cyclone IV FPGA on the Terasic DE2-115 board. We estimate the FPGA resource utilization of the design and also perform power analysis of the same.

4.1 Design

This section describes the send & receive instructions format and the execution flow of these two instructions.

4.1.1 User Defined Instruction

The special2 opcode of the MIPS Instruction Set Architecture allows us to implement up to 16 UDIs. Two new instructions namely SEND and RECEIVE are added to the existing MIPS instruction set. Figures 4.1 and 4.2 below depict the Send and Receive instruction formats respectively.

OPCODE[31:26]	SRC[25:20]	DEST[19:6]	FUNC[5:0]
011100			011000

Figure 4.1: Send Instruction Format

OPCODE[31:26] 011100	SRC[25:20]	DEST[19:6]	FUNC[5:0] 010101

Figure 4.2: Receive Instruction Format

4.1.2 Send

Basic functionality of the send instruction is to move data from one MIPSfpga processor to other through Hoplite. Figure 4.3 shows the data flow model for send instruction. UDI module of the MIPSfpga core will wait for instruction valid signal from instruction decoder and then reads data from the communication RAM's location whose address is specified as a part of the instruction. Then it generates NOC packets and hands it over to the Hoplite (NOC), only if the network is free otherwise it buffers packets into the FIFO instead of stalling execution of the next instruction to avoid processor overhead. Packets stored in FIFO are transferred when network becomes free. If FIFO is full processor is asked not to issue new send instructions by asserting stall signal.

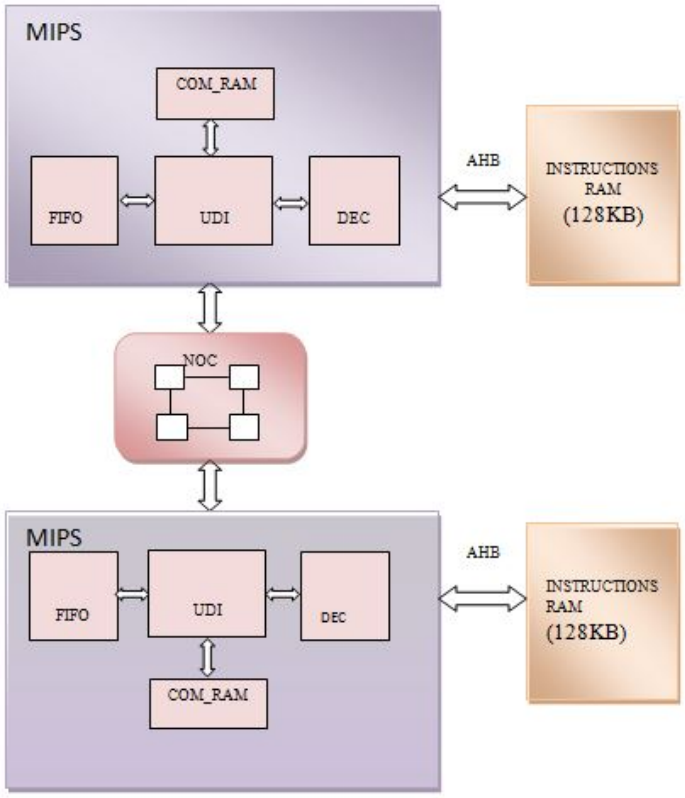

Figure 4.3: Send instruction flow

4.1.3 Receive

Upon completion of send instructions execution receive instructions are executed by the processor to update the register file of the receiver MIPSfpga processor with the data received from the other MIPSfpga cores. UDI at the receiver core will check for the valid packets from NOC and buffer them in another internal scratch pad RAM. Once NoC finishes sending data, the receive UDI is enabled to load registers file with the received data. We don't provide synchronization at the hardware level to intimate the processor about received data. It has to be taken care at the compiler level.

4.2 Performance Improvements

MIPSfpga comes with a 2 way set associative cache. It helps in improving injection rate of packets from 0.1 (with cache disabled) to 0.5. MIPSfpga lacks instruction pre-fetching mechanism which is a drawback in terms of throughput. Whenever the processor encounters a cache miss the controller fetches 4 instructions from memory and during this time, the execution unit will go to idle state which leads to 50% reduction in injection rate. To counter this issue we pad previously executed instructions in the 4 cycle gap in order to achieve an injection rate exactly equal to 1.

4.3 Methodology

We use Altera ModelSim Starter Edition 10.0 to simulate our design and verify its functionality. We write **tcl** scripts to automate the simulation process. This tcl script is responsible for compiling verilog files, libraries and generating input vectors. Figure 4.4

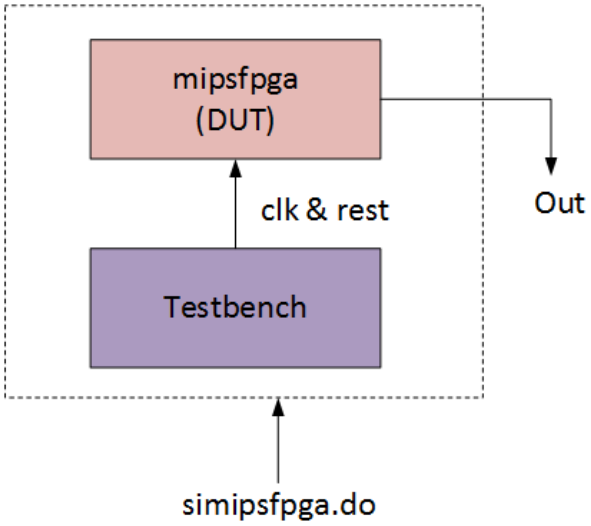

Figure 4.4: Multiplication Simulation

depicts the test setup of our project. There are two phases involved in testing. In the first phase, instruction RAM configuration files are loaded with machine codes for send/receive instructions are generated for individual MIPSfpga cores. In the second phase, modelsim project is opened through tcl script and simulation is initiated.

./mips.tcl MODE REPEATITION PATTERN RATE

MODE This parameter specifies the number of MIPSfpga cores to be instantiated. There are seven modes, from MODE0 to MODE6 each corresponding to 4, 8, 16, 32, 64, 128 and 256 MIPSfpga processor cores respectively.

REPEATITION This parameter is used to generate packets in large number. If specified as 1, for 4 MIPSfpga cores it generates 4 x 32 packets where 32 is the number of scratchpad register locations which is fixed. 2x number of packets can be generated by making this parameter 2.

PATTERN This argument is used to specify the type of network traffic pattern to be used in UDI packet generation. Our design is capable of generating three different types of traffic patterns like RANDOM, LOCAL and TORNADO by passing values of 1, 2 and 3 respectively to this parameter.

RATE This argument is used to select the injection rates between 0.5 and 1.

Code generation Addition of new instructions makes it impossible for the compiler to generate a machine code with new user defined instructions. We write a TCL script to facilitate the purpose of machine code generation. It generates individual instruction RAM configuration file for each MIPSfpga core.

Simulation During simulation each MIPSfpga core executes initialization machine code to initialize its resources i.e, Data cache, Instruction cache and TLB before executing send/receive instructions. At the end of simulation various log files are generated such as mips_bw.csv, fifo_depth.csv, data_error.txt and fifo_overflow.txt. These log files are used to analyze the throughput, data correctness and FIFO depth sensitivity of the system.

4.4 Results

We perform simulation under various system sizes, synthetic workloads, injection rates and fifo depths. In this section, we show sustained rate, throughput and FIFO depth sensitivity tests results for the various design configurations. We also tabulate resource and power utilization data for the modified design.

Injection/Offered rate : The rate at which packets are injected into the network.
Sustained rate : The maximum rate that can be delivered by the network for a given injection rate. Sustained rate is always less than or equal to the injection rate.

4.4.1 Sustained Rate Test

Sustained rate test is performed for 2, 4, 8, 16, 32, 64, 128 and 256 core systems at two different injection rates 0.5 and 1 for local and random traffic patterns. sustained rate data for local and random patterns under various system sizes and injections rates is shown in Figure 4.5. From the plots it can be noticed that as the system complexity increases sustained rate of the system decays. Local patterns have relatively higher sustained rate as compared to other two patterns because, each core sends packet to an other core which is located within the specified window size but in case of random, the destination address is completely random.

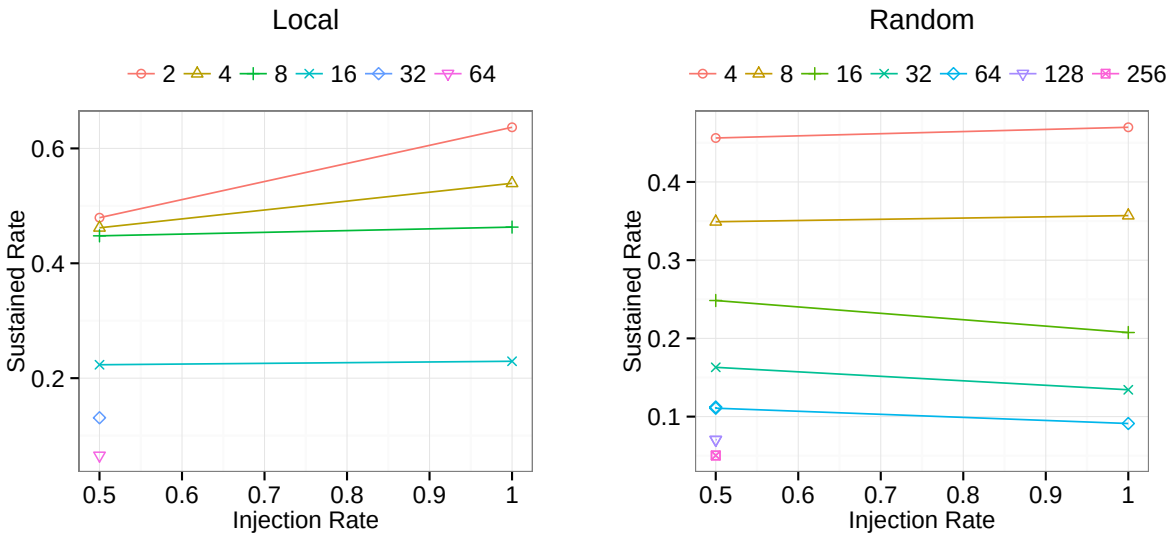

Figure 4.5: Sustained Rate v/s Injection Rate

4.4.2 Throughput Measurement

Figure 4.6 shows the throughput analysis plots for random and tornado patterns at an injection rate of 0.5. From the plots it is evident that the bandwidth is directly proportional to the number of cores and is the highest for 256 cores in the case of random pattern which is 2x that of whatever is achieved with the tornado pattern. But for number of cores up to 8, Tornado scores over Random because of it's higher sustained rate.

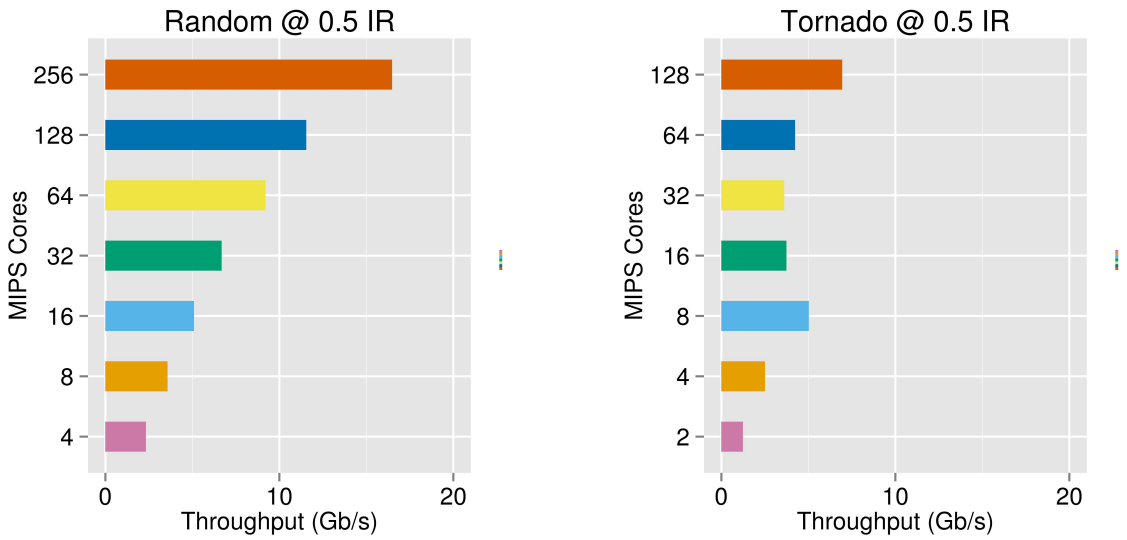

Figure 4.6: Throughput v/s Number of cores

4.4.3 Sensitivity Test

FIFO depth sensitivity test is performed to observe minimum fifo depth required for the given system to avoid stalling of processor. This test is executed for 4, 8, 16 and 32 core systems at 0.5 injection rate and the results are shown in Figure 4.7. Our evaluation results clearly show that FIFO depth is directly proportional to the instruction execution rate. For a system with 4 cores, fifo with 32 locations is more than sufficient to avoid stalling where as a 32 core system requires a fifo of depth 256. In case of 32 core system a fifo of depth 32 results in fetching instruction once in 5 clock cycles instead of two clock cycles. This test proved that to have a better execution rate for a larger system we need larger fifo.

4.4.4 Area Utilization

Table 5.1 indicates the area utilization of a single core MIPSfpga system and a comparison between the original and modified MIPSfpga processors. We use Altera Quartus 14.1 tool to record the resources utilization. After adding the UDI instructions, there has been a significant rise in the number of logic elements and registers utilized because of an extra UDI logic added for the send and receive instructions and also the increase in 1 M9k block is because of the FIFO and scratchpad RAMs.

4.4.5 Power measurements

Table 4.2 depicts the power measurements of a single core MIPSfpga system and a comparison between the original and modified MIPSfpga processors. We use PowerPlay

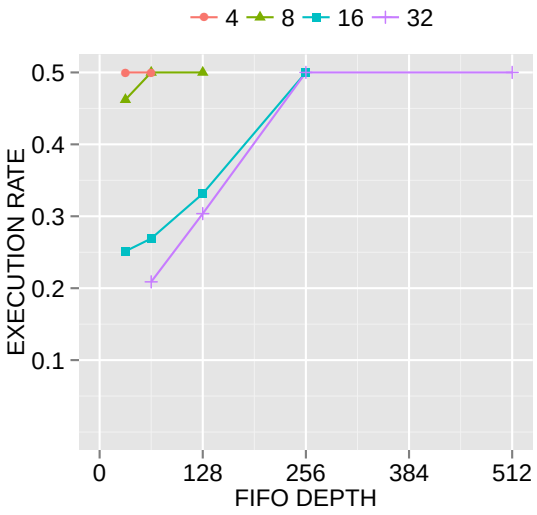

Figure 4.7: Execution rate v/s FIFO depth

Table 4.1: Resource Utilization

Resources	Original	Modified
Total Logic Elements	15,326 / 114,480 (13%)	15,377 / 114,480 (13%)
Total Registers	7,425 / 117,053 (6%)	7,467 / 117,053 (6%)
M9k blocks	403 / 432 (93%)	404 / 432 (94%)
Embedded multipliers	0 / 532 (0%)	0 / 532 (0%)
Total PLLs	1 / 4 (25%)	1 / 4 (25%)
Total memory bits	3,223,936 / 3,981,312 (81%)	3,224,000 / 3,981,312 (81%)
Total pins	57 / 529 (11%)	57 / 529 (11%)

Power analyzer tool of Altera Quartus 14.1 to analyze the power consumption. The overall power dissipation in the modified design is a little higher than the original one because of the extra UDI logic which rises the switching activity.

4.5 Conclusion

In this project we show how to implement send and receive instructions to exchange data in a DSM multiprocessor system. We conclude that the smaller multiprocessor systems result in a better sustained rate and the execution rate is directly proportional to the FIFO depth. We show that the worst case latency increases with the complexity of the system. Our design consumes a little more area and less power as compared to the original design.

Table 4.2: Approximate Power Dissipation

Power consumption	Original	Modified
Total Thermal Power Dissipation	511.65mW	515.04mW
Core Dynamic Thermal Power Dissipation	351.56mW	354.91mW
Core Static Thermal Power Dissipation	104.58mW	104.62mW
I/O Thermal Power Dissipation	55.51mW	55.51mW

Chapter 5

Optimized MIPSfpga Soft Processor

We aim at optimizing the existing MIPSfpga soft processor in order to map 256 such cores along with the NoC interconnect onto a high end Altera or Xilinx FPGA. We can use this optimized MIPSfpga core to design a parallel multi core processor overlay interconnected with Hoplite NoC. Optimized processor is still be a general purpose processor as original design with no data and instruction cache support.

5.1 Original Architecture

Figure 5.1 depicts a single instance of the original MIPS soft processor design along with its various components in the form of a block diagram. The Execution Unit, Instruction Decoder and System Co-Processor mainly make up the Execution Data Path (EDP) which are used to generate the virtual addresses to access the shared memory. The Memory Management Unit (MMU) converts these virtual addresses into physical addresses. The Instruction Cache and Data Cache represent the memories which provide faster access to instructions and data respectively. The Bus Interface Unit BIU) controls the transfer of data between the MIPSfpga soft processor and the shared memory. The Multiplication and Division Unit (MDU) performs the arithmetic operations for the processor.

We will now explain the optimization performed in each of the components of the original design:

5.1.1 Execution Data Path

The EDP was modified to directly generate the addresses of the instructions in the Instruction RAM which was added by us. The instructions from the I-RAM are then passed on to the Instruction Decoder for decoding. The decoded instructions are sent

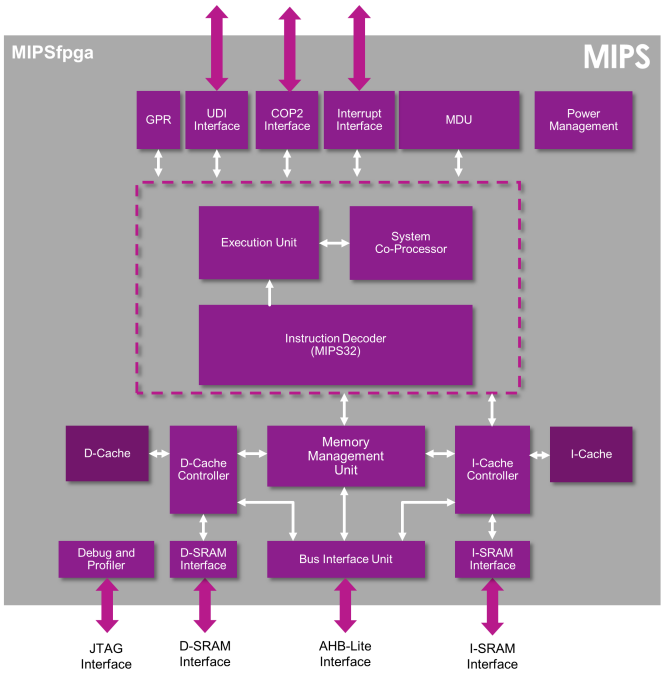

Figure 5.1: Original MIPS Architecture

back to the EDP either for arithmetic operations or to access the UDI for transfer of data between different processors.

5.1.2 Memory Management Unit (MMU)

As we are directly generating the address of instructions to be accessed in the I-RAM, we can completely bypass the MMU which used to perform the task of converting virtual addresses to physical addresses previously. It is a redundant component in the new design and is ,therefore, not synthesized by the tool.

5.1.3 Instruction Cache and Data Cache Controllers

Since we wanted to modify the existing architecture into a Distributed shared-memory multiprocessor (DMP), we completely eliminate the I-Cache and D-Cache. However, we cannot completely eliminate their controllers as it generates some critical signals which are required by the Instruction Decoder.

5.1.4 User Defined Instructions

A User Defined Instruction (UDI) unit with different sections for sending and receiving instructions across processors using the user-defined Send and Receive instructions is im-

plemented by us. Each processor has its own scratchpad allowing them to independently access their personal memories and also share data between processors.

5.1.5 Bus Interface Unit (BIU)

The removal of the external shared memory allows us to eliminate the BIU which used to control the transfer of data across the shared memory between processors. However, due to certain critical signals which were being generated by the BIU, we could not completely eliminate it. We have managed to remove all of its redundant functionality but kept certain critical signals like indicating to the EDP when to generate the next address.

5.1.6 Register File

The original design implemented the Register File in the form of registers, each of which was 32-bit wide. In order to save out on the number of registers utilized, we replaced the RF with an overclocked RAM with 2 read ports and one write port. This RAM works at a frequency which is 2x faster than the original circuit frequency. It has been implemented using a Block-RAM with a size of 9 kbits meaning that it can store 256 32-bit instructions.

5.1.7 Multiplication and Division Unit (MDU)

The MDU implemented multiplication using a series of additions in accordance with Booth's Algorithm. We replace this by a DSP block to reduce the number of utilized LUTs. Also support for division was removed from the original implementation. DSP blocks have also been utilized to implement special functions of the processor such as Multiply-Add and signed and unsigned operations.

5.1.8 Master Pipeline Control (MPC)

The MPC is basically the instruction decoder which gets instructions from the Instruction RAM, decodes them and passes on the decoded instructions to the EDP for execution.

5.2 Optimized Architecture

Based on all the optimizations performed by us, the architecture is as represented in Figure 5.2. We have not been able to completely eliminate certain components due to some critical signals which they were generating and were essential for the operation of the processor on a whole.

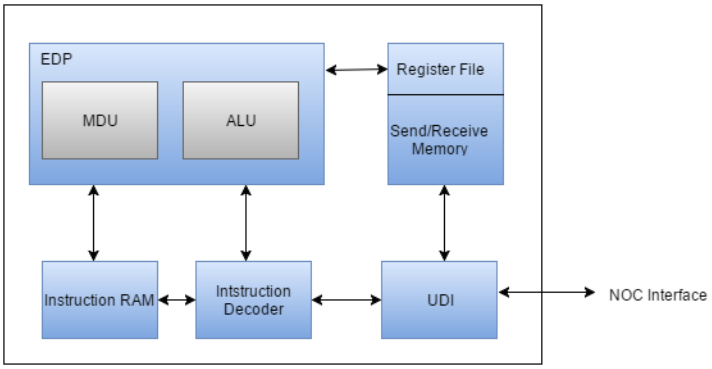

Figure 5.2: Optimized MIPS core

5.3 Synthesis

We use Altra Quartus 14.1 to target the Cyclone IV E device. The tool provides us information regarding the resource utilization of the design as well as the minimum clock period required for operation/ the maximum operating frequency. Table 5.1 shows the resource utilization of original and optimized design. From the table we can observe that, there is $\approx$ 70% reduction in number of logic elements as compared to the original.

Table 5.1: Resource Utilization

Component	Original (No. of Logic Elements)	Optimized (No. of Logic elements)	Percentage Reduction
MMU	3692	0	100%
CPZ	1699	1406	17.25%
RF	1043	104	90%
MDU	762	251	67.1%
EDP	3239	1837	43.3%
MPC	993	862	13.2%
BIU	919	20	97.8%
DCC	1707	57	96.7%
ICC	910	55	94%
Total	14884	4592	69%

Chapter 6

Hierarchical Torus Networks on Chip

Hierarchy can help us improve the worst case routing latencies of deflection torus NoCs. The low complexity of deflection routing make them particularly well-suited for low-cost implementations of modern FPGAs. However, when compared to classic buffered NoCs, the cost of deflections severaly impacts the worst-case routing latencies for certain packets.We show how to design a deadlock-free multi-layer torus NoC that can lower the worst case deflection costs by as much as 2–3×. By providing a parallel escape channel in the lower layers, we guarantee deadlock-free routing in the hierarchical network while imposing no operational constraints on the network clients. We generate layouts for implementing the multi layer NoC on the ML605 board (XCV6LX240T FPGA), VC707 board (XC7VX485T FPGA) and large multi-die XC7V2000T chips while consuming 10–15% of FPGA LUT resources. For instance with the 16×16 NoC, we reduce worst case deflection costs by 1.5–10× while simultaneously improving sustained rates by 1.5–2× and lowering energy requirements by 1.5–2× for a range of statistically-generated traffic patterns.

6.1 Idea

In the proposed design, traffic is split into two levels (1) a local sub-network that routes local traffic and (2) a global upper-level network that helps in longer distance communication. This restructuring of the network-on-chip (NoC) into multiple level improves latency behavior of the flat 2D deflection torus. The two levels of network still have 2D unidirectional buffer less deflection routing.

The main idea behind this reorganization is to isolate local traffic to multiple local regions of NoC. As traffic in other local regions will not be interfered with local traffic, the overhead and penalty (extra distance packet has to cover due to deflection) per

deflection in routing will be reduced. Global traffic (packets/messages whose destination is in another local network) will traverse both levels of network. The fact that less packets/messages will be travelling through global layer as compared to a flat 2D deflection torus means a reduction in traffic in global level. Only global traffic will traverse both networks and will suffer less congestion and associated penalties in the upper level network.

6.2 Multi Layer Network Design

In this section, we describe the structure of our multi-layer network design. Fundamental idea of multi- layer network implementation is to lessen number of deflection counts by splitting a large network into small- scale networks referred as local network and establish communication among them, with the aid of global network and the client link interface. Every node (e.g., CPU, cache slice, or memory controller) resides on one local network, and connects to one node router on that network. There is no buffering or flow control within any local network; packets are buffered only in pipeline registers. Local networks are connected to one or more levels of global network via client link interface to form a multi-layer hierarchy. A client link has a set of transfer FIFOs (one in each direction) between the local networks. A typical 4X4 multi-layer network is shown in figure 6.1.Here the 4×4 single layer network is bifurcated into four 2×2 local networks and connected them through 2×2 global network.

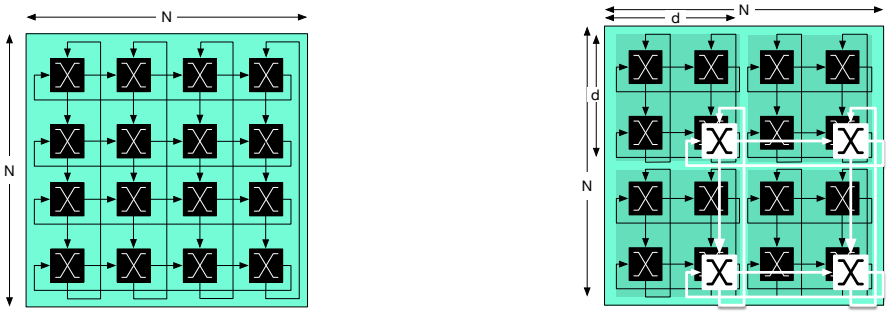

Figure 6.1: Comparing Flat 2D Torus with Multi-Layer NoCs. Overall system size $N \times N$, while lower-level is $d \times d$.

6.2.1 Node Router

At each node, we place a single node router, shown in figure 6.2. A node router is very simple: it passes through circulating traffic, allows new traffic to enter the network through a MUX, and allows traffic to leave the network when it arrives at its destination. Each router contains one pipeline register for the router stage, so the router latency is exactly one cycle. Each node router has three input(North,West and Client) and two output (East and South) ports.

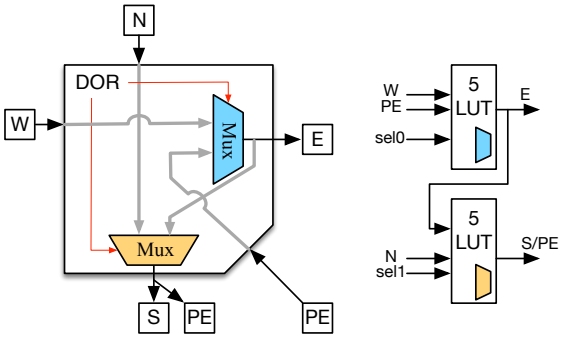

Figure 6.2: High-Level diagram of Hoplite router microarchitecture. Mux mapped to fractured Xilinx 5-LUTs. Processing Element (PE) implements user logic that injects and receives network traffic.

6.2.2 Client Link

Client link plays a main role in conveying packets from local network to global network and vice versa. It has set of transfer FIFOs (one in each direction) as shown in figure 6.3. Client link consumes packets that require a transfer, only if the respective transfer FIFO has space. The packets available in a transfer FIFO are injected into the network upon receiving bus free signal from the network. When a packet requires a transfer but the respective transfer FIFO is full, the packet deflect in its current network and try again next time it encounters the destination ready signal from the respective client link.

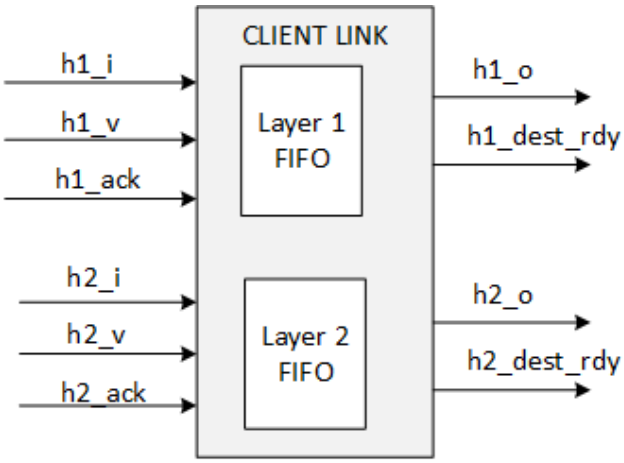

Figure 6.3: Client Link Interface

6.2.3 Multi-Channel Global Network

To tackle issue of reduction in bandwidth we use multiple channels at global network. Number of channels at global layer is equal to the size of local layer (2x2 local layer means 2 channels at global layer). The selection of channel by client link is done in a round robin way to ensure that network does not get clogged at other client links. During selection if the round robin selected channel is not free then any empty channel is selected to route the packet from Layer1 FIFO. Every channel has it's own Layer2 FIFO. One packet (taken out from the selected Layer2 FIFO) is injected into local network by client link. The FIFO selection is also done in round robin way. This ensures none of packets is stuck in Layer2 FIFO.
Multiple channels helps in parallelizing traffic at global network thereby reducing the number of cycles taken to route all packets.

6.3 Deadlock Free Network

6.3.1 Deadlock Condition

By using deflections rather than buffering and blocking flow control to manage packet transfers within the local network, multi-layer design retains node router simplicity,unlike buffered torus designs. This change comes at the cost of potential deadlock (if packets are forced to deflect forever). We introduce mechanism that implements two channel local network as shown in figure 6.5, to provide a deterministic guarantee of deadlock-free operation.To understand the deadlock state, let us consider an example as shown in figure 6.4,there are two packets (one coming from North and other from West end) in local network (say L1) whose destination resides in another local network (say L2) , at the same time two more packets are heading towards L1 from global network. If the Layer1 and Layer2 FIFOs are full at this point of time, then the client link interface will not welcome packets from either of the layers, so packets are forced to deflect in their current network. Now the deadlock occurs. Packets are neither able to enter into FIFOs because they are already at full capacity and nor FIFOs are able to inject packets in any of the layers because both H1 and H2 nodes are busy routing/deflecting other packets. Increasing the FIFO size will only delay the occurrence of deadlock rather eliminating it completely.

6.3.2 Two Channel Local Network

In order for the system to operate in deadlock free state, the local-global network interconnect must guarantee that every packet is eventually delivered to its destination. Multi-layer design ensures deadlock free network using two-channel local network. The

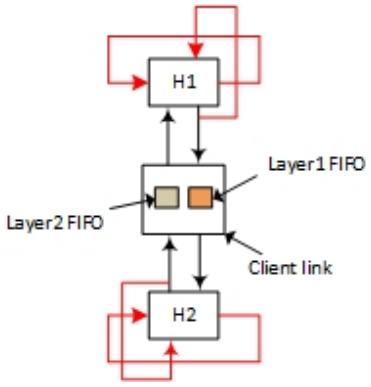

Figure 6.4: Deadlock condition

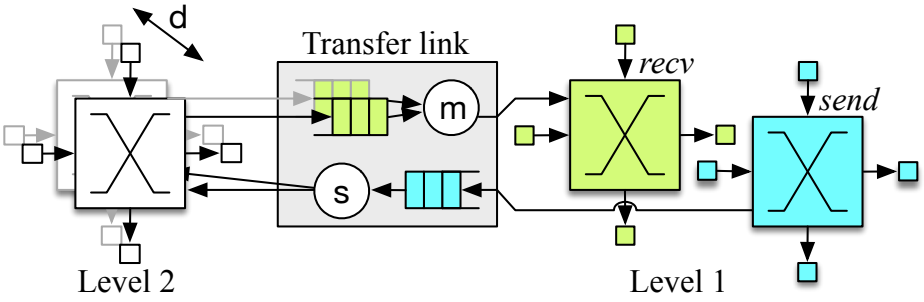

Figure 6.5: Deadlock Free Interface design

mecha- nism is shown in figure 6.6. While one channel is reserved for packets to be sent out of the local network & to perform internal routing, other for receiving packets from global network. One channel serves as input while other serves as output port for client link. Both channels are able to send packets to client but client is able to inject packets only in channel1. Channel1 routes packets internally and transfers packets (meant for other local network) to client link. On the other hand, channel2 accepts packets from client link and routes them to destination client within the local network. At any point of time channel2 in local network will be only responsible for receiving packets. This ensures that local network will always be able to receive new packets from global network. As global network is able to send packets out it will also be able to receive packets. Thus, ensuring that design is deadlock free.

6.4 Methodology

We perform simulations under various system sizes, synthetic workloads, injection rates and hierarchical arrangements along with physical mapping experiments on modern Xilinx FPGAs. This lets us evaluate metrics such as achieved bandwidths, worst case latencies, physical resource requirements and operating frequencies.

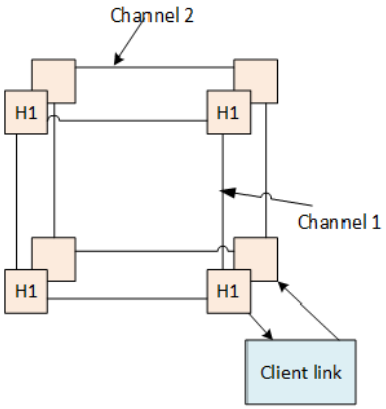

Figure 6.6: Two channel local network

6.4.1 Simulation

We perform RTL simulations of the various network configurations. Our performance and energy data is obtained directly from synthesizable RTL implementation of the switch. We implement packet injection logic to support various network traffic patterns while also correctly modeling source queueing delays. We also develop automated analysis tools that post-process simulation logs to confirm correctness of the NoC operation as well as permit a detailed analysis of performance. As the design space of possible combinations is large, we use the open-source **iverilog** tool and cheap Google Compute Engine resources to run 16-32 parallel instances of the simulation on one machine. We explore various combinations of N, d, synthetic pattern [15], and injection rates as listed in Table II. We sample the lower injection rates more as performance generally saturates above 50% injection rates and most realistic multi-processor workloads offer 510% injection rates [17].

Table 6.1: Design-space Exploration of NoC parameters.

Parameter	**Range**
N	4×4, 8×8, 12×12, 16×16
d	2×2, 3×3, 4×4
Pattern	`LOCAL`, `RANDOM`, `BITREV`, `TRANSPOSE`, `TORNADO`
Sigma (`LOCAL`)	1,2,4,8,16
Injection Rate	1%, 5%, 7%, 10%, 15%, 20%, 50%, 100%

In figure 6.7, we show different conventional static traffic patterns. In this project we use Random, Bit Reverse, Transpose, Tornado and Local traffic patterns.

6.4.2 Physical Layout

We use Xilinx ISE 14.7 and Vivado 2015.4 to target the ML605 (Virtex-6) and VC707 (Virtex-7) devices respectively. We generate UCF and XDC constraints to precisely

Name	*Pattern*
Random	$\lambda_{sd} = 1/N$
Permutation	
Bit permutations	
Bit complement	$d_i = \sim s_i$
Bit reverse	$d_i = s_{b-i-1}$
Bit rotation	$d_i = s_{i+1} \bmod b$
Shuffle	$d_i = s_{i-1} \bmod b$
Transpose	$d_i = s_{i+b/2} \bmod b$
Digit permutations	
Tornado	$d_x = s_x + (\lceil k/2 \rceil) \bmod k$
Neighbor	$d_x = s_x + 1 \bmod k$

Figure 6.7: Conventional Static Traffic Patterns [10]

layout the NoCs on the FPGA. We use XPower (ISE/Virtex-6) and Vivado Power Analysis/Estimation (Virtex-7) tools to calculate power usage. When using folded layout for the 2D torus, we consistently achieve clock frequency of 280–330MHz for various NoC configurations. This frequency is close to the typical operating frequency of the logic and interconnect fabric of the Virtex-6 FPGA. For the large FPGA devices such as the Virtex-7 2000T, the chip is composed of four separate dies, called SLRs super logic regions in the FPGA CAD flow, connected by interposers. For these designs,crossing dies is slower (limited at 400MHz with aggressive pipelining) and consumes more power. Layouts and NoC sizing for large FPGAs must minimize inter-SLR crossing to NoC links. For the XC7V2000T quad-die FPGA, we have 13K interposer connections that support larger data widths up to 512b. We are able to fit 16X16 NoCs with a 4X4 subnetwork (see Figure 6.17) within the die constraints of the XC7V2000T chip.

6.5 Results

We perform various experiments and provide an evaluation of our proposed multi-layer network design against single layer design. A mult-layer network yields significant performance advantages over a single layer network when the offered throughput is high and the local network size is small enough. Multi-layer network improves performance 2x times for LOCAL traffic pattern compared to a single layer network. The deflection count is reduced by up to 5 times for LOCAL traffic pattern whereas for RANDOM traffic pattern we see reduction in deflection count by 3 times at small offered throughput. As most of the deflections are in local layer deflection length/cost is also reduced.

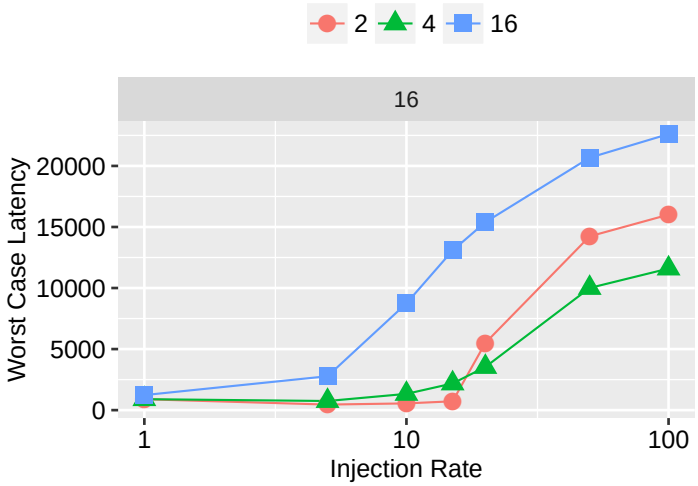

Figure 6.8: Worst case latency vs. offered throughput for 16x16 system with variable local network size and local traffic pattern

6.5.1 Throughput Tests

Under randomly generated communication workloads (uniform random), the NoC design is able to sustain throughput only up to a certain input offered throughput (≈ 0.35) for 4x4 and (≈ 0.08) for 16x16 systems with local network size equal to 2, as shown in figure 6.9. This expected behavior is due to congestion effects in the NoC at high traffic loads and the complexity of the system. In all scenarios, the multi-layer network with least local network size (2x2), sustained throughput is approximately 35% more than that of single layer network at higher offered throughput. This gap is the result of less number of deflections in multi-layer network. Sustained throughput of single layer network dominates multi-layer network at lower offered throughput, but the difference is minimal.

Apart from uniform random pattern, we evaluated the multi-layer design across other commonly-used synthetic routing patterns as shown in figure 6.10. In this case, the LOCALITY-based pattern performs well against other traffic patterns all the time. Up to a certain input offered throughput (≈ 0.05) all the traffic patterns except BITREV resulted in same sustained throughput. At higher offered throughput , LOCAL pattern's sustained throughput is thrice that of other traffic patterns and this is true only in multi-layer networks. The BITREV pattern generally route poorly on all networks with negligible sustained throughput.

With Locally generated communication workloads, the NoC design could achieve sustain throughput of (≈ 0.4) for 4X4 and (≈ 0.15) for 16X16 systems with local network size equal to 2, as shown in figure 6.11. In contrast to RANDOM, multi-layer network

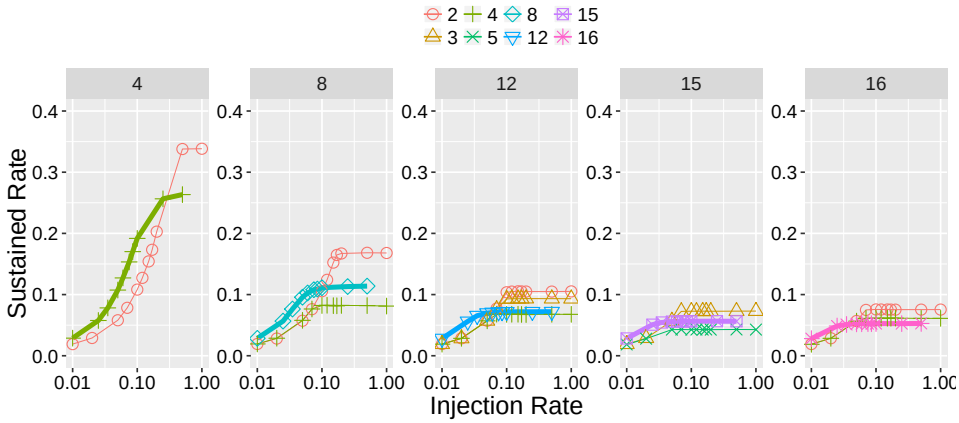

Figure 6.9: Sustained vs. Offered Throughput in the NoCs for random traffic pattern (each point is an offered throughput)

with LOCAL traffic pattern has clear win against single layer network irrespective of local network size. Interesting property of local pattern is that, it ensures 2 times the sustained throughput of RANDOM pattern at higher offered throughput. This property is valid only for multi-layer networks. The space between sustained throughput of LOCAL and RANDOM pattern is minimal (≈ 0.01) for single layer networks.

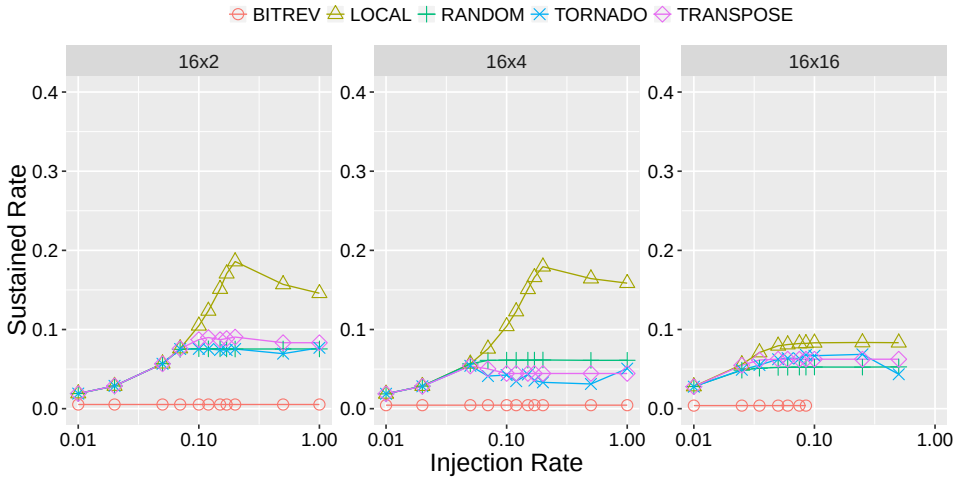

Figure 6.10: Sustained vs. Offered Throughput in the NoCs across various traffic patterns (16X16 with variable local network size, each point is an offered throughput)

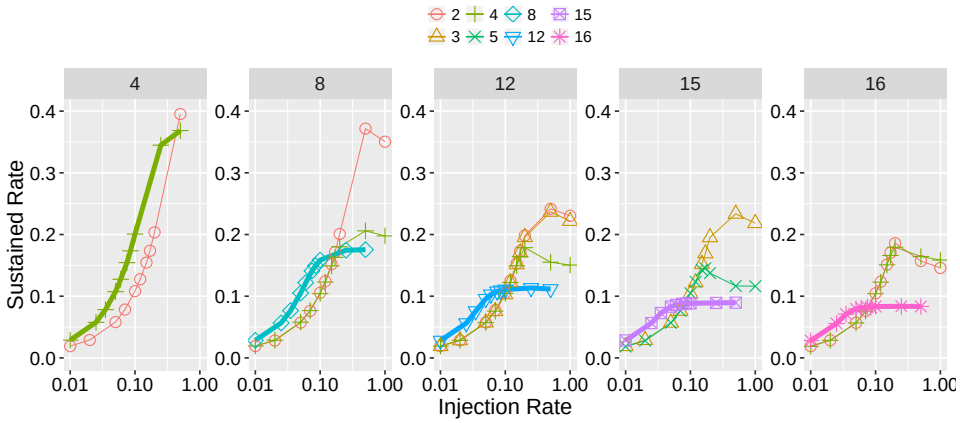

Figure 6.11: Sustained vs. Offered Throughput in the NoCs for local traffic pattern (each point is an offered throughput)

6.5.2 Latency Analysis

Figure 6.12 illustrate density histogram of packet latency for a LOCAL traffic at an offered throughput=0.1. Multi-layer design is able to restrict the worst-case packet latency to $\approx 1K$ cycles for an offered throughput=0.1 with local traffic pattern and local network size of 2 & 4 on a 16x16 system. On the other side, the single layer implementation (local network size=16) allow longer worst case routing delay of $\approx 10K$ cycles under identical workload conditions. This out-turn is as a consequence of higher deflection count in a single layer network as discussed in deflection count analysis section. As local network size increases, peak of the density plot shift towards the right side.This implies that, multi-layer network is capable of conveying more number of packets in short period as compare to single layer network.
At an offered throughput of 1, packet latency for 16x16 system with LOCAL traffic pattern is approximately 50 times worse than packet latency at 0.01 offered throughput.System performance degrades as the offered throughput increases due to more congestion in network. This effect is depicted in figure 6.13.

6.5.3 Deflection Count Analysis

In figures 6.14 - 6.16, we measure deflection at local and global network for multi and single layered systems as a function of offered throughput. Increase in local network size results significant increase in deflection count. This is because small sized networks has less number of packets. The cost of deflections in multi-layer network is much lesser than in single layered network. This outcome is as a result of multi-layer network prioritizing the convey of packets which are local to the network by buffering non local packets (packets whose desired port is in other local network) at the client interface. Multi-layered network offers smaller deflection length for a given packet as compared to single

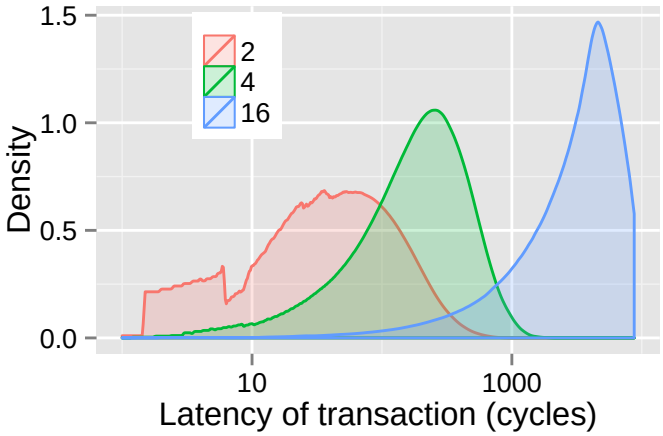

Figure 6.12: Density Histogram of Packet Latency for 16x16 system (variable local network size) NoC routing local traffic, offered throughput=0.1

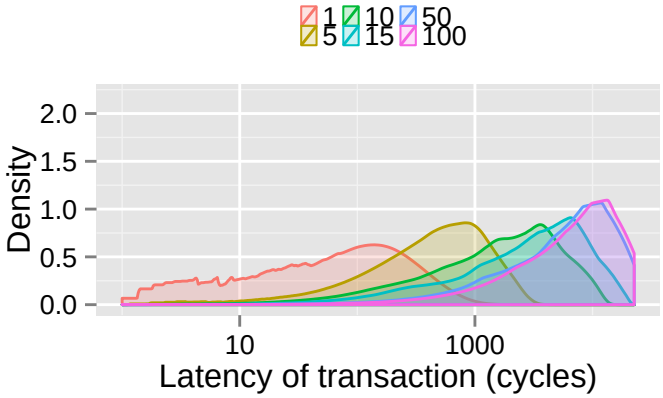

Figure 6.13: Density Histogram of Packet Latency for 16x16 system NoC routing local traffic for various offered throughput

layered network as an advantage of it's hierarchical design.
Multi-layer design is 3 times better (in terms of deflection count) than single layer up to certain offered throughput (≈ 0.08) for RANDOM pattern. At offered throughput=0.10 deflection count decreases by a factor of 5 for LOCAL pattern. For LOCAL pattern the number of deflections at local network is more than that for RANDOM pattern because for a random pattern more number of packets are sent out of the local networks.

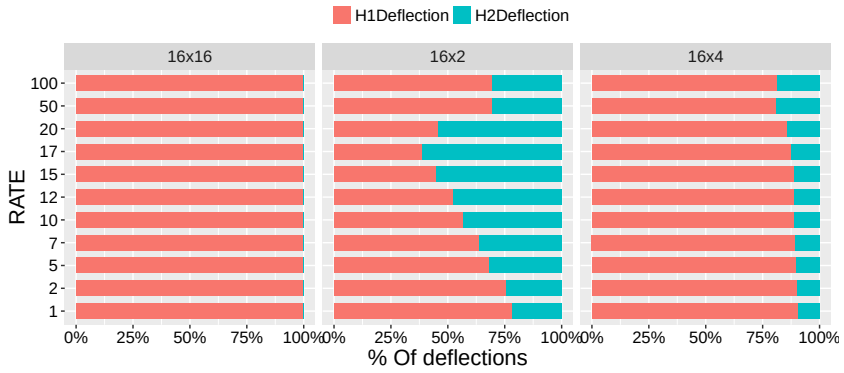

Figure 6.14: Percentage deflection at local and global network vs. offered throughput for local traffic pattern

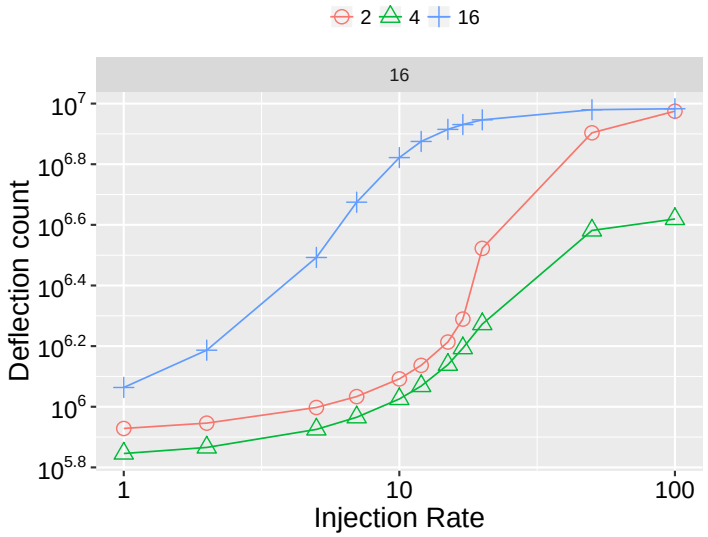

Figure 6.15: Total deflection vs. Injection rate for 16x16 system with variable local network sz and local traffic pattern

6.5.4 Physical Implementation

We generate folded floorplans for FPGA devices that span complete chip as shown in Figure 6.17 for the large multi-die XC7V2000T FPGA. Even 16×16 NoCs consume less than 15% of the chip resources for the NoC infrastructure leaving most of them for user logic. It is evident that with careful folded layout we are able to achieve 490 MHz speeds for 16×16 NoCs while exceeding 300 MHz+ frequencies for all configurations. Typical FPGA designs run at around 250-300 MHz implying the regular layout of the NoC is unlikely to affect overall system critical path delay and fast speeds of the flat single-level NoCs offer no particular advantage. Furthermore, is it clear that the addition of hierarchy

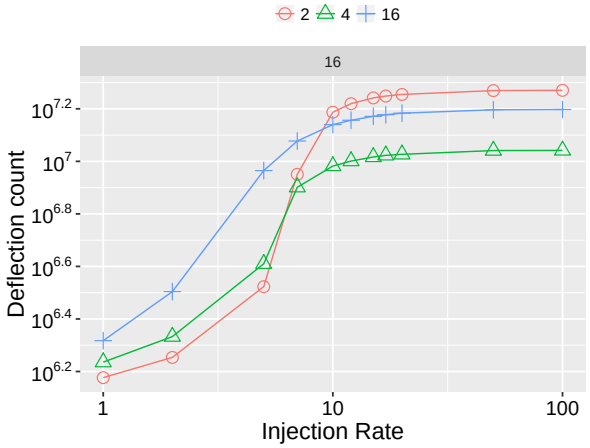

Figure 6.16: Total deflection vs. offered throughput for 16x16 system with variable local network size and random traffic pattern

increases cost only marginally 1–5% of chip area.

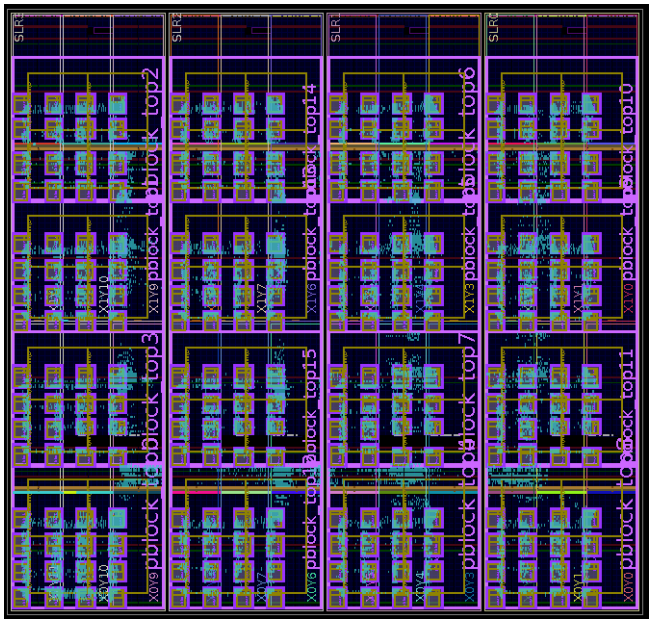

Figure 6.17: 16×16 (4×4 lower level) 320 MHz chip-spanning 32b NoC on the Xilinx XC7V2000T (multi-die FPGA).

References

[1] D. J. William and B. Towles,*"Route packets, not wires: On-chip interconnection,"* DAC, 2001.

[2] Kevin Andryc, Murtaza Merchant, and Russell Tessier, *FlexGrip: A Soft GPGPU for FPGAs*, Field Programmable Technology (FPT), 2013 International Conference, 9-11 Dec. 2013

[3] M. Nickray, M. Dehyadgari and A. Afzali-kusha, "Power and Delay optimization for network on chip,"*Proceedings of the 2005 European Conference on Circuit Theory and Designs, Cork, Ireland,* pp. 273-276, 2005.

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services,"*Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on,* pp. 13-24, 2014

[5] M. Palesi, R. Holsmark, S. Kumar and V. Catania, "Application specific routing algorithms for network on chip," *IEEE Transactions on Parallel and Distributed Systems,* pp. 316-339, 2009.

[6] M. K. Papamichael and J. C. Hoe, "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs,"*the ACM/SIGDA international symposium,* p. 37, 2012.

[7] Y. Huan and A. DeHon, "FPGA optimized packet-switched NoC using split and merge primitives,"*Field-Programmable Technology (FPT), 2012 International Conference on,* pp. 47-52, 2012.

[8] J. Gray, "GRVI-Phalanx: A Massively Parallel RISC-V FPGA Accelerator," *3rd RISC-V Workshop,* 2016.

[9] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs,"*Field Programmable Logic and Applications (FPL), 2015 25th International Conference on,* pp. 1-8, 2015.

[10] B. P. Towles and B. Dally, Principles and Practices of Interconnection Networks, San Francisco, CA, USA: Morgan Kaufmann Publishers, 2004

[11] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys.,* pp. 1-51, 2006.

[12] G. Michelogiannakis, D. Sanchez, W. J. Dally and C. Kozyrakis, "Evaluating Bufferless Flow Control for On-chip Networks," *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium,* pp. 9-16, 2010.

[13] R. Ausavarungnirun, C. Fallin, X. Yu, K. K.-W. Chang, G. Nazario, R. Das, G. H. Loh and O. Mutlu, "Design and evaluation of hierarchical rings with deflection routing," *Computer Architecture and High Performance Computing (SBAC-PAD), no.* 2014 IEEE 26th International Symposium, p. 230237, 2014.

[14] K. Chapman, "Saving Costs with the SRL16E," [Online].Available: http://www.xilinx.com/support/documentation/white_papers/wp271.pdf.

[15] P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente and J. . Gregorio, "TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers," *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on,* pp. 99 - 106, 2012.

[16] R. Holsmark, S. Kumar, M. Palesi and A. Mejia, "A methodology for deadlock free routing in hierarchical networks on chip," *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on,* pp. 2-11, 2009.

[17] P. Gratz and S. W. Keclker. Realistic Workload Characterization and Analysis for Networks-on-Chip Design. In *The 4th Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI),* 2010.

[18] B. S. Landman and R. L. Russo, "On a pin versus block relationship," *Computers, IEEE Transactions on,* pp. C-20(12):14691479, 1971.

[19] http://www.ijircce.com/upload/2015/june/186_Performance_new.pdf.