Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1-1-2012

Digital Fabric

Sudheer Goshi

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Goshi, Sudheer, "Digital Fabric" (2012). *Dissertations and Theses*. Paper 115.

<https://doi.org/10.15760/etd.115>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Digital Fabric

by

Sudheer Goshi

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Dan Hammerstrom, Chair
Douglas V. Hall
Christof Teuscher

Portland State University
©2012

ABSTRACT

Continuing advances with VLSI have enabled engineers to build high performance computer systems to solve complex problems. The real-world problems and tasks like pattern recognition, speech recognition, etc. still remain elusive to the most advanced computer systems today. Many advances in the science of computer design and technology are coming together to enable the creation of the next-generation computing machines to solve real-world problems, which the human brain does with ease. One such engineering advance is the field of neuromorphic engineering, which tries to establish closer links to biology and help us investigate the problem of designing better computing machines.

A chip built with the principles of neuromorphic engineering is called as neuromorphic chip. Neuromorphic chip aims to solve real-world problems. As the complexity of the problem increases, the computation capability of these chips can become a limitation. In order to improve the performance and accomplish a complex task in the real-world, many such chips need to be integrated into a system. Hence, efficiency of such a system depends on effective inter-chip communication. Here, the work presented aims at building a message-passing network (Digital Fabric) simulator, that integrates many such chips. Each chip represents a binary event-based unit called spiking analog cortical module. The inter-chip communication protocol employed here is called as Address Event Representation.

Here, the Digital Fabric is built in three revisions, with different architectures being considered in each revision. The complexity is increased at each iteration stage. The experiments performed in each revision test the performance of such configuration systems and results proves to lay a foundation for further studies.

In the future, building a high level simulation model will assist in scaling and evaluating various network topologies.

Acknowledgements

First and foremost, I offer my sincerest gratitude to Dr. Dan Hammerstrom, who has supported and advised me throughout my thesis. I would like to thank him for providing me an opportunity. I attribute the level of my Masters degree to his patience, encouragement, and effort, and without him this thesis, too, would not have been completed or written.

I would like to thank Dr. John Lynch, Professor Roy Kravitz for their help during my work. I am also grateful to the committee members, Dr. Douglas V.Hall and Dr. Christof Teuscher for their constructive comments on this thesis.

I would like to express my gratitude and love to my parents, sister, brother for their constant support and encouragement. I would like to express my thanks to Srinivas Rao Vanka, Renjith, Shilpa Kulkarni who as good friend were always willing to help and give their best suggestions. Lastly, many thanks to Pradeep, Mandar, Rao Arun, Vikram and the complete motley crew for their support.

Contents

Abstract	i
Acknowledgements	iii
List of Tables	x
List of Figures	xi
1 Introduction and Related Work	1
1.1 Neuromorphic Engineering	1
1.2 DARPA SyNAPSE Program	2
1.3 Main Objective of the Thesis	3
1.4 Thesis Organization	4
2 System Architecture Overview And FPGA Feasability Study	5
2.1 Chapter Overview	5
2.2 System Architecture Overview	5
2.3 Address Event Representation	8
2.4 Assumptions	10
2.5 FPGA Feasibility Study	11

3	Rev0 DF Architecture	18
3.1	Rev0 DF Architecture Overview	18
3.2	Key Assumptions In Rev0 DF Architecture	20
3.3	Rev0 DF System Diagram	22
3.4	Address Event Representation Format: Rev0 DF Architecture . . .	24
3.4.1	Different Modes Of Operation	24
3.5	Rev0 DF Architecture Sub-Module Descriptions	25
3.6	Pulse_Generator	25
3.7	APP_Out	26
3.8	APP_In	27
3.9	Bus_Control	28
3.9.1	Sub-Module Input/Output Signals:	29
3.10	Rev0 DF Architecture Timing Diagram	30
3.11	Rev0 DF Architecture Summary	32
4	Rev1 DF Architecture	34
4.1	Rev1 DF Architecture Overview	34
4.2	Ambric Registers:	36
4.2.1	Pseudo-code for Ambric Registers:	37
4.3	Rev1 Micro-architecture System View One Dimension One Direction	39
4.4	APP_Router Sub-Module Micro-architecture	40
4.5	Clock domains in the Rev1 DF Architecture	41
4.6	Address Event Representation Format: Rev1 DF Architecture . . .	42
4.6.1	Different Modes Of Operation	42
4.7	Key assumptions in Rev1 DF architecture	46

4.8	Rev1 DF Architecture Sub-Module Descriptions	48
4.9	Pulse_Generator or APP	48
4.10	Packet_Generator or AER/Convert	48
4.11	APP_Router	49
4.11.1	Sub-Modules In APP_Router	49
4.12	Rev1 DF Architecture Summary	52
5	Rev2 DF Architecture	54
5.1	Rev2 DF Architecture Overview	54
5.2	Ambric Registers	57
5.3	Rev2 APP_Router Sub-Module Micro-architecture	58
5.4	Clock domains in the Rev2 DF Architecture	59
5.5	Address Event Representation Format: Rev2 DF Architecture . . .	59
5.5.1	Operation	59
5.6	Assumptions	64
5.7	Rev2 DF Architecture Sub-Module Descriptions	66
5.8	Pulse_Generator or APP	66
5.9	Packet_Generator or AER/Convert	67
5.10	APP_Router	67
5.10.1	Sub-Modules In APP_Router	67
6	Modelsim And FPGA	71
6.1	Chapter Overview	71
6.2	Modelsim	71
6.3	Approach in implementing Rev0 DF Architecture	72
6.4	Address Event Representation Packet Format	73

6.5	Approach in implementing Rev1 DF Architecture	74
6.6	Approach in Implementing Rev2 DF Architecture	75
6.7	Xilinx Spartan 3E FPGA Board	75
6.8	Microblaze RISC processor	76
6.9	Implementation on the FPGA Board Using Microblaze	77
6.10	GPIO: General Purpose Input/Output	78
6.10.1	PLB Interface Module	79
6.10.2	Interrupt Control	80
6.10.3	GPIO Core	80
6.11	UART: Universal Asynchronous Receiver Transmitter	80
6.11.1	Functional Description	80
7	Results	82
7.1	Chapter Overview	82
7.2	Statistics Definitions	82
7.3	Modelsim Simulation Results	83
7.4	Rev0 DF Architecture Results: Operation Mode: Go-to	83
7.4.1	Rev0 DF Architecture Simulation results: Varying Packet Generation Rate	85
7.5	Rev1 DF Architecture Simulation Results	87
7.5.1	Address Event Representation Format: Rev1 DF Architecture	87
7.5.2	Rev1 DF Architecture Results: Operation Mode: Go-to . . .	88
7.5.3	Rev1 DF Architecture Simulation Results: Varying The Packet Generation Rate	89
7.5.4	Rev0 and Rev1 DF Architectures Result Comparison	91

7.6	Rev2 DF Architecture Simulation Results	92
7.7	Address Event Representation Format: Rev2 DF Architecture . . .	92
7.7.1	Rev2 DF Architecture Results: Operation Mode: Go-to . . .	93
7.7.2	Rev2 DF Architecture Simulation results: Varying Packet Generation Rate	94
7.7.3	Rev0, Rev1 and Rev2 DF Architectures Results Comparison	96
8	Conclusion & Future Work	99
8.1	Conclusion	99
8.2	Future Work	100
	References	102
	Appendices	104
A	Rev2 APP_Router Sub-Module Input/Ouput Signals Description	105
A.1	Signals Description used in Figure 5.2 and 5.4	105
A.1.1	Figure 5.2 Signals	105
A.1.2	Figure 5.4 Signals	105
B	Rev0, Rev1 & Rev2 Modelsim Simulation Waveforms	108
B.1	Rev0 Results: Arbitration model	108
B.1.1	Broadcast Mode	110
B.1.2	Goto Mode	111
B.1.3	Rev1 Broadcast Mode	116
B.1.4	Rev1 Goto Mode	119
B.1.5	Rev2 Broadcast Mode	121

B.1.6	Rev2 Goto Mode	124
-------	--------------------------	-----

List of Tables

3.1	Address Event Representation Format In The Rev0 DF Architecture	25
4.1	Address Event Representation Format In The Rev1 DF Architecture	43
5.1	Address Event Representation Format In The Rev2 DF Architecture	59
6.1	Address Event Representation Format For Rev0, Rev1, and Rev2 DF Architecture	73
7.1	Revisiting Address Event Representation Format In The Rev1 DF Architecture	87
7.2	Revisiting Address Event Representation Format In The Rev2 DF Architecture	93

List of Figures

2.1	Digital Fabric Architecture Overview	6
2.2	Simple Shared Bus Address Event Representation System	8
2.3	Address Event Representation Protocol	9
2.4	Structure of an FPGA	14
2.5	Xilinx Spartan 3E Logic Cell Conceptual Diagram	14
2.6	FPGA Design Flow Chart	16
3.1	Digital Fabric Interface With APP	19
3.2	Rev0 DF Architecture System Diagram	23
3.3	Content Addressable Memory In The Rev0 DF Architecture	28
3.4	Rev0 DF Architecture APP_Out Timing Waveform	30
3.5	Rev0 DF Architecture APP_In Timing Waveform	31
3.6	Rev0 DF Architecture Bus Timing Waveform	31
4.1	Digital Fabric Architecture Rev1 System Diagram	35
4.2	Ambric Registers: Interface between the nodes in Rev1	36
4.3	Rev1 Micro-architecture System Diagram	39
4.4	Rev1 APP_Router Sub-Module Micro-architecture	40
4.5	Content Addressable Memory In The Rev1 DF Architecture	52
5.1	Digital Fabric Rev2 DF Architecture System Diagram	55

5.2	Digital Fabric Rev2 APP_Router Sub-Module Connectivity	56
5.3	Rev2 APP_Router Sub-Module Micro-architecture	58
6.1	Microblaze Processor : RISC 32 bit Soft Core Processor from Xilinx	77
6.2	Rev1 DF Architecture on FPGA	78
6.3	GPIO Software IP by Xilinx	79
7.1	Rev0 Simulation Results Arbitration vs. Non-Arbitration Model . .	84
7.2	Rev0 Simulation Results Go-to Mode : Lower Packet Generation Rate	85
7.3	Rev0 Simulation Results Go-to Mode : Higher Packet Generation Rate	86
7.4	Rev1 Simulation Results Go-to Mode: Lower Packet Generation Rate	89
7.5	Rev1 Simulation Results Go-to Mode: Higher Packet Generation Rate	90
7.6	Comparison between Rev0 and Rev1 DF Architecture	91
7.7	Rev2 Simulation Results Go-to Mode: Lower Packet Generation Rate	94
7.8	Rev2 Simulation Results Go-to Mode: Higher Packet Generation Rate	95
7.9	Rev0, Rev1 and Rev2 DF Architectures Comparison plot	96

Chapter 1

Introduction and Related Work

Over six decades, modern electronics has evolved through a series of significant developments leading to the advancement of microprocessor technology. Current microprocessors have higher computation capabilities and help in solving complex problems. The tasks such as pattern recognition, speech recognition, and sensory integration cannot be effectively addressed by today's most advanced distributed computer systems, but they are easily done by the human brain. Hence, we need to use the insight from the human information-processing system and design better computing machines. Neuromorphic Engineering [1] is a new and emerging interdisciplinary field that tries to establish closer links to biology and ultimately contributes to the design of better computing machines.

1.1 Neuromorphic Engineering

Carver Mead introduced the concept of Neuromorphic engineering[1] in the late 1980's. Neuromorphic engineering proposes to replicate the architecture of the neural circuits on silicon chips, emulating neurons and the synaptic connections between them. Neuromorphic systems are a possible bridging technology between biological neural network models and VLSI hardware. The chips built using the

principles of neuromorphic engineering use an array of similar processing elements. These elements could be either analog, digital or mixed-signal [2] components and interact in a non-linear manner with each other to model a neural network [3].

The motivation for building neuromorphic chips is to solve real-world problems. As the complexity of the problem increases, the computation capability of these chips can become a limitation. Despite rapid advances in the scaling of VLSI technology, which promises to provide more transistors on a chip, it is not possible to integrate the full functionality of a complete nervous system on a chip. Hence, to achieve this functionality and to improve performance, many such chips need to be integrated in a single system. As a result, the performance of such a system highly depends on efficient inter-chip communication rather than on the system clock operational frequency [4][5].

1.2 DARPA SyNAPSE Program

SyNAPSE (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) is a DARPA (Defense Advanced Research Projects Agency) funded project, which aims to build a new kind of computer based on neural circuits. The motivation behind the program is to “investigate innovative approaches that enable revolutionary advances in neuromorphic electronic devices and are scalable to biological levels.” The SyNAPSE program seeks to define a new path forward for creating useful, intelligent machine. These new machines will excel at the kinds of distributed, data-intensive algorithms required to solve real-world problems. The systems are implemented in hybrid CMOS/nano-grid technology at a very large scale (100 millions of neurons). The three contractors for the program are IBM, HP and HRL.

HRL is responsible for creating a chip called the Asynchronous Pulse Processor (APP). The APP implements basic analog spiking neuro circuits. At Portland State University, we are responsible for a digital network called as Digital Fabric (DF), which integrates many such APPs into a system and is responsible for communication between APPs through spikes (timed binary events).

1.3 Main Objective of the Thesis

The aim of the thesis is to build a message-passing network (Digital Fabric), which integrates spiking analog cortical modules (APPs). The inter-APP communication protocol employed is called Address Event Representation (AER) [4]. The Mead Lab proposed AER in the year 1991 for transferring spikes between bio-inspired chips [5][6][7]. In the HRL system the APP represents the cortical column. An APP takes asynchronous spikes as input and produces asynchronous spikes as output. Likewise APPs communicate internally and externally via Spikes. The Digital Fabric (DF) is responsible for translating external spikes into AER, which is routed via the DF to the addressed APPs. The computation model is asynchronous and the DF can be implemented asynchronously, but the current APP is synchronous, so we have implemented the DF as a synchronous system. The arriving AER packets are converted into pulses and are used as input to the other neighboring APPs. The DF architecture is modular, and the architecture provides scalability, predictability and design reuse. The design process consisted of 3 iterations of the architecture in increasing complexity.

1.4 Thesis Organization

The introductory chapter is followed by Chapter 2, which provides an overview of the DF architecture, the preliminary architecture development, and the FPGA feasibility study. Chapter 3 discusses the Rev0 DF architecture, which uses a single, synchronous, shared arbitrated bus. Chapter 4 presents the Rev1 DF architecture which is synchronous, one-dimensional token ring architecture [8]. Chapter 5 discusses the Rev2 DF architecture, which, is an extension of the Rev1 DF architecture to a 2D torus packet network [9]. Chapter 6 presents the simulation environment used to implement the architectures. Chapter 7 discusses important observations and results for the various architectures. Chapter 8 concludes the work and outlines future research.

Chapter 2

System Architecture Overview And FPGA Feasability Study

2.1 Chapter Overview

This chapter provides an overview of the system architecture and the FPGA feasibility study. The implementation assumptions and FPGA design flow are also summarized.

2.2 System Architecture Overview

The basic system architecture is shown in the figure 2.1. In the architecture, defined here, the neuron is at the lowest level, which is based on a spiking model [10] [11] and is implemented in the APP by custom analog CMOS. Figure 2.1 shows the hierarchy in an overly simplistic model of the visual cortex [12] and the corresponding hardware system. The hardware system is based on the fundamental Address Event Representation (AER), which was developed almost 20 years ago by Carver Mead's group at Caltech [4].

Figure 2.1: Digital Fabric Architecture Overview

Figure 2.2 shows a simple shared bus AER system. When a spike arrives at an APP output signal, it is converted into an address in the memory of the DF node that interfaces to the APP. The address gives the destination APP, for the arriving pulse as well as the corresponding signal in the destination APP. The DF node then sends a packet containing the address to the addressed APP, which generates a pulse on the addressed line.

A virtual connection is defined by the addresses in the routing table. After a connection is established between two nodes, a spike stream may be delivered between nodes. Here, by using “virtual connections”, which are defined by addresses loaded into the bus access units, the packet network integrates APP’s into a “virtual” connectivity graph, which is normally a 2-dimensional sheet, and the sheets into higher-level structures. These virtual networks are implemented by the connectivity patterns defined by address mappings at the DF-APP interfaces, as well as whatever connectivity is defined internal to the APP. An objective of the DARPA SyNAPSE program is to implement neurobiological inspired circuitry at the nano-scale and then integrate these structures into complex systems.

Figure 2.2: Simple Shared Bus Address Event Representation System

2.3 Address Event Representation

Neuromorphic chips use Address Event Representation (AER) for external communication. AER is event driven and is an asynchronous communication protocol used for transferring spikes between the chips. Generally, events are communication packets that are sent from a sender to one or more receivers. Mahowald and Sivilotti proposed the idea of the AER to transmit spikes from an array of neurons on one chip to a connected neuron in the array of the second chip [5] [6] [7]. A neural network [3] consists of numerous basic cells, which are either transmitting or receiving data [13].

The basic idea of AER is that each cell in the network generates a spike when its

internal state has crossed a threshold value; its address is then transmitted onto a common bus. Each time a cell or sender device generates a spike, it communicates with the array periphery and a digital word representing a code or address is placed on the external inter-chip digital bus. In the receiver chip, the spikes are directed to the connection specified by the address. In this way, cells with the same address in the sender and receiver chips are virtually connected by spike streams. Additional handshaking lines (Acknowledge and Request) are used for completing the asynchronous communication. In this thesis, the Digital Fabric (DF) connects all the APP chips via AER packets. The DF is responsible for the transmission of AER packets over the channels to the addressed APP. When an AER packet reaches the destination APP, it is decoded, and a spike is generated on the addressed line.

Figure 2.3: Address Event Representation Protocol

The DF was developed in several stages. This thesis includes the Rev0, Rev1,

and Rev2 DF architectures. The Rev0, Rev1, and Rev2 DF architectures are implemented in Verilog using the Modelsim simulation environment. For implementation in Verilog, spikes are also referred to as pulses in this thesis. The Rev1 DF architecture is synchronous and uses a 1-dimensional token ring network [8], for inter-APP packet communication. For the Rev2 DF architecture, the synchronous Rev1 DF architecture is retained, except that it grows to include a 2-dimensional torus packet network [9]. The Rev1 DF architecture was synthesized on an FPGA board (Xilinx Spartan 3E student kit [14]). Xilinx’s soft-core processor Microblaze is used as an interface between DF nodes/modules and output terminals such as the on-board LCD and the PC host (Hyper-Terminal) to display results.

2.4 Assumptions

The research here focuses on building a simple message-passing network (Digital Fabric) and communication within the network for both 1-dimensional and 2-dimensional routing. The routing algorithm is kept simple (shortest routing algorithm) and does not address any new routing algorithm or make any comparison with existing routing techniques. For the work discussed here, basic architectures are implemented. The Digital Fabric is built in three revisions, with different architectures being considered in each revision. The complexity is increased at each iteration stage. For the DF prototypes discussed above, AER packets are dropped, greatly simplifying the protocols and buffer size requirements. The experiments performed in each revision tests the performance of such configuration systems and results prove to lay a foundation for further studies. For further revisions, these assumptions will be addressed. In conclusion, correction or detection is not

included in my FPGA prototyping.

2.5 FPGA Feasibility Study

Functional Verification of the chip plays an important role in the life cycle of chip manufacturing. It is the key for reducing development costs and increasing the time to market. The three widely used approaches for functional verification are simulation, emulation and FPGA based prototyping. Simulations imitate the architectures with the help of simulators. A software simulation refers to an event-driven logic simulator that operates by propagating input changes through a design until a steady state condition is reached. Simulation is the key activity in the design verification process. The incorrect behavior in the design can be found early in the design life cycle with the help of simulation. For verification, the entire design is simulated with the help of the state of the art simulators such as ModelSim, PSPICE or LTSpice. A simulator being implemented in software is relatively slow and it is not feasible to completely test, via simulation, all possible states and inputs of any non-trivial system.. Also, the need to find the hardware bugs as early as possible in the design cycle drives the emulation and FPGA prototyping effort. Emulation is the process of mapping an entire design into a hardware emulation platform designed to further increase verification performance. For example, hardware accelerated emulation platforms like Palladium from Cadence and Velcore from Mentor. Hardware based emulations, being the better alternative to verify the design in a real time environment, still have the disadvantage that the clock cycle is one or two orders of magnitude slower than the actual hardware. The third method of verification, known as FPGA prototyping, is popular since

FPGA implementations can operate at speeds of 50 MHz-100 MHz, which are comparable to ASICs. The FPGA prototype refers to the construction of custom hardware or the use of reusable hardware to construct a hardware representation of the system. FPGAs have been used as system modeling environments due to the fact that they provide a large number of gates, competitive clock rates and simple implementation. With advances in silicon and design tools, larger designs can be fitted into FPGAs. With the advent of on-board logic analyzers and real time RTL debugging, the debugging technology now matches the speed of FPGAs.

For this thesis, Modelsim is used as an environment to perform the simulations and verify the functionality of the architecture. Taking into account the flexibility provided by FPGAs, the next step in our design process was to prototype the design on a FPGA. Here, Rev1 DF architecture is prototyped onto FPGA. In order to study the possibility of FPGA prototyping, several issues had to be addressed. The most important issues were as follows:

How can we feed the test vectors (AER packets) into the design and how can we read and analyze the results of prototyping?. In order to address these issues, a few assumptions were made. The general methods for test vector patterns generation, can be either deterministic, or pseudo-random or hybrid. Here, a pseudo-random test pattern generator was used for this purpose. The random test vectors are generated using Linear Feedback Shift Registers (LFSR). Generally, the LFSR is a shift register whose input bit is driven by the Exclusive-or (XOR) of some bits of the overall shift register value. These patterns generated using LFSR have all the desirable properties of random numbers, but are algorithmically generated by the hardware pattern generator and are therefore repeatable. To address the issue of reading and analyzing the results, a Microblaze RISC Processor, an IP

core provided by Xilinx, was used to interface the architecture to a host PC, and to pass results to a Hyper-terminal and the LCD. For the architectures presented here, a clock (50 MHz) provided by the Xilinx Starter Kit [14] was used as input to the design.

An FPGA consists of a two-dimensional array of programmable logic blocks, which may include general logic, memory and multiplier blocks. These are surrounded by a programmable routing fabric that allows blocks to be interconnected. The array is surrounded by input/output (I/O) pads, which connect the chip to the board [15]. A custom design can be implemented by specifying the HDL, which then synthesizes the cell function. After completion of design and synthesis, we can download the desired logic cell and obtain the custom circuit. A logic cell contains a configurable combinational circuit and a D- Flip Flop (D-FF). Generally the configurable combination circuit is implemented as a Look Up Table (LUT). An example of a three input LUT implementation of $A \text{ xor } B \text{ xor } C$ is shown below in the figure 2.4. Memory blocks, I/O interface circuits, and clock management circuits are all implemented as macro cells that are designed and fabricated at transistor levels. A Xilinx Spartan 3E [14] is used to prototype the design.

Figure 2.4: Structure of an FPGA

Figure 2.5: Xilinx Spartan 3E Logic Cell Conceptual Diagram

Each Spartan 3E device logic cell consists of a four input LUTs and a D-FF as shown in the figure 2.5. In addition, the logic cell also contains a carry circuit and a multiplexing circuit, which are used to implement arithmetic units and wide multiplexers. In Xilinx devices, two logic cells are grouped to form a slice, and four slices are grouped to form a configurable logic block. The simplified design flow of an FPGA-based system is as shown below in the figure 2.6. The left portion of the flow defines the refinement and programming process while the right portion of the flow is the validation process, where functional verification of the system is performed.

Figure 2.6: FPGA Design Flow Chart

Below are the major steps in the design flow:

- Design the system using the Verilog or VHDL or System C.
- Develop a test bench in HDL and perform RTL simulation.
- Perform synthesis and implementation. During synthesis, software transforms the HDL language to generic gate-level components such as latches, flip flops, & gates. The implementation process consists of three processes, namely: translate, map, and place and route.
- The translate process merges the multiple design files to a single netlist. During the mapping process, generic gates in the netlist are mapped to FPGAs logic cells and Input/Output buffers. Placement and routing is responsible for deriving the physical layout inside the FPGA chip. After placing the logic cells and routing them to connect to various signals, static timing analysis is performed, which determines the various timing parameters, such as the maximum propagation delay and the maximum clock frequency.
- Finally, a bit file is generated according to the final netlist and this file is downloaded to the FPGA device to realize the functionality of the design.

Chapter 3

Rev0 DF Architecture

3.1 Rev0 DF Architecture Overview

In the basic Rev0 Digital Fabric (DF) architecture a node consists of a single, synchronous, arbitrated, multi-access bus and a set of bus access units. The bus is 8 bits wide and sends 16-bit AER packets in two clock cycles. The two main types of bus access units are: the APP_In and the APP_Out. APP_In reads the AER packets off the bus, and converts them into input pulses to the APP. APP_Out takes pulses from the APP and converts them into AER packets, which are transmitted on the bus. This functionality is shown in figure 3.1. In essence, the DF should be invisible to the APPs such that pulses are routed with some delay and minimal jitter noise from the source APP's output wires to the destination APP's input wires. The signal delays do not have to be identical, but they should be reasonably consistent. That is, each wire sees the same delay for each spike it sends with small random delays (jitter noise). However, it is possible that due to efficient network digital packet processing, all delays may be much smaller than the normal neuron processing times. For implementation in Verilog, a node is divided into four main sub-modules: the Pulse_Generator, the APP_Out, the APP_In, and the Bus_Control and are detailed in section 3.5.

Figure 3.1: Digital Fabric Interface With APP

The Rev0 DF architecture is a simple and limited implementation of the DF. It is synchronous and uses a single shared bus. The basic design consists of a set of sub-modules that implement the bus, spike generation, and reception; there are two bus access units in each node, one for the input and one for the output. The number of bus access units of each type in the system is variable, but is limited to 32. Since the number of wires connected to each bus access unit is fixed at 32, a total of 1024 wires (32×32) can be connected via the bus, thus forming a system with 1024 APP inputs and 1024 APP outputs. The bus is 8 bits wide and sends 16-bit AER packets in 2 clock cycles. Each 16 bit packet consists of a 10-bit address, where 5 bits specify the destination bus access unit

and 5 bits specify the wire within the unit. There is also a Bus_Control sub-module (for initializing the bus and addresses) and a stochastic pulse generator for each APP_Out sub-module. The systems can either be arbitrated, where shared communication resource conflicts cause packet delays, or non-arbitrated, where such conflicts cause packet loss or corruption. Because we use a multiplexor based bus, there cannot be packet corruption, but we can lose AER packets. For testing, we have implemented two bus access models.

In the arbitration model, all APP_Out units send a request. Those APP_Out units that are not granted access to the bus keep trying with a particular packet until access is granted. In the non-arbitrated model, the multiplexor structure guarantees that only one unit that has the highest priority accesses the bus in the arbitration cycle. However, all requests assume their packets were delivered. So, the AER packets from lower priority units are lost. Experiments by Boahen [2] have shown that arbitrated systems have better performance. At the same time, this result depends significantly on the general algorithms and what packet loss means at the level of algorithm function. For arbitrated systems, more than one packet can be generated while waiting to access the bus. Other design issues include the size of the buffer, and the method to handle buffer overflow (delete the oldest packet or newest, etc.). For the Rev0 DF architecture presented here, the AER packets are buffered (buffer size of 8 and 16 are used for simulation purposes) and if another packet is generated, it overwrites the first (the oldest is lost).

3.2 Key Assumptions In Rev0 DF Architecture

This section summarizes various architecture and implementation assumptions:

- The correct operation is assumed to be the delivery of a pulse from an APP's output wire to its destination APP's input wire.
- Ideally, the time delay for each pulse should be constant, and pulses should not be lost.
- The initial pulse generation rate is one pulse per wire (assuming each wire is a neuron output) per second. However, we can increase the spike generation rate by several orders of magnitude so that we can study congestion issues.
- As, the pulse rate increases, we see two kinds of conflict: simultaneous pulses arriving at a single APP_Out, and simultaneous APP_Out sub-modules attempting to access the bus. During system operation, we collect statistics on the results of this congestion and packet delivery. If pulses arrive at the same time at an APP_Out or while the sub-module is waiting to put a previous pulse packet on the bus, the arriving pulses are dropped. As discussed above, these could be queued, but to keep Rev0 simple we do not do that here.
- The APP_Out bus arbitration is switchable. If it is enabled, then sub-modules that are waiting simultaneously for bus access will continue to wait until they are granted bus access. If bus arbitration is disabled, then the multiplexed bus only allows the highest priority sub-module data access to the bus, the AER packets from other sub-modules are lost.

Next a general system overview is presented, and then the individual sub-modules are described.

3.3 Rev0 DF System Diagram

A basic system with four bus access units (2 in and 2 out) is shown in the figure 3.2. The “Reset” and “Clock” signals go to all the sub-modules and are not shown in the figure. The Sys_Control is a test bench used to initialize the system, which includes loading the CAMs in the APP_In sub-modules, and controlling the system operation. The Sys_Control unit gives commands to each Pulse_Generator to start generating the pulses, and the Sys_Control controls the rate at which the pulses are generated.

Figure 3.2: Rev0 DF Architecture System Diagram

3.4 Address Event Representation Format: Rev0 DF Architecture

3.4.1 Different Modes Of Operation

- 000000 No-Op.
- 000001 Go-to mode: transmit spike to destination wire in destination unit.
- 000010 Broadcast mode: Broadcast spikes to destination wire in all units.

In the basic Rev0 Digital Fabric (DF) architecture a node consists of a single, synchronous, arbitrated, multi-access bus and a set of bus access units. Bits 15-10 in the AER packet determine the mode of operation, which are listed above. If the value is 000000, no operation is performed; if the value is 000001, the mode is set to Go-to; and if the value is 000010, Broadcast mode is set. In Go-to mode, the AER packet is sent from a single bus access unit to a specified destination bus access unit (i.e., there is just one sender, and one receiver) and the unit address in the AER packet contains the destination bus access unit address. In Broadcast mode, it is assumed that every bus access unit in the node (connected via the single, synchronous, arbitrated bus), receives the AER packets (i.e. there is just one sender, but the information is sent to all connected receivers). In Broadcast mode, the unit address in the AER packet (i.e. 9:6 in the AER packet, as shown in table 3.1) contains the source bus access unit address (i.e. the source which generated the AER packet). The unit address and the wire address in the 16 bit AER packet, are formed with the help of a pseudo random number generator using a LFSR. The mode of operation (i.e. the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of the Modelsim simulation. The AER packet formation is the same for Rev1 and Rev2 architectures. In the

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 3.1: Address Event Representation Format In The Rev0 DF Architecture

Rev0 DF architecture, the Bus_Control sub-module decodes the information in the AER packet, while in the Rev1 and the Rev2 architectures the router sub-module is responsible for decoding the AER packet information and performing the requested action. In Broadcast mode, the Bus_Control sub-module transmits an AER packet to all the bus access units in the node. In Go-to mode, again the Bus_Control sub-module is responsible for decoding the AER packet, and then transmitting it to the destination bus access unit.

3.5 Rev0 DF Architecture Sub-Module Descriptions

A node consists of following four main sub-modules: the Pulse_Generator, the APP_Out, the APP_In, and the Bus_Control.

3.6 Pulse_Generator

The Pulse_Generator generates spikes for all 32 APP_Out sub-modules. Each APP_Out sub-module has a 32 bit input; hence the output of the Pulse_Generator is Pulse_out [1023:0] (32 * 32). Pulses are active for one clock cycle. The Pulse_Generator has 2 modes of operation, Random and Sequential mode, generating pulses at regular intervals. In Sequential mode, only 1 bit out of 1023 bits is asserted in sequence. In Random mode, a random number of pulses appear randomly at random locations at the output of the Pulse_Generator sub-module. For random pulse generation, the \$Dist_normal (mean, standard deviation) function

is used. The function returns a random integer value at every clock cycle. This integer value forms the inter-spike interval in the Random mode. Thus, by varying the attributes, the mean and the standard deviation, the inter-spike interval is set and is random, but in the Sequential mode `$Dist_normal` function is not used to set the interval between spike generation. The interval between the spikes generation is a settable parameter for Sequential mode and can be varied by the user at the beginning of the simulation. The Pulse_Generator generates pulse for all 32 APP_Out sub-modules. Each APP_Out sub-module has a 32 bit input; hence the output of the Pulse_Generator is Pulse_out [1023:0] ($32 * 32$). Pulses are active for one clock cycle. The Pulse_Generator can operate in either Random or Sequential mode, generating pulses at regular intervals. In Sequential mode, only 1 bit out of 1023 bits is asserted in sequence. In Random mode, pulses appear randomly at random locations. For the generation of random pulses in Random mode, the `$Dist_normal` (mean, standard deviation) function is used. The function returns a random integer value at every clock cycle. This integer value forms the inter-spike interval in the Random mode. The interval between spikes is a settable parameter only for Sequential mode. Thus, by varying the attributes, the mean and the standard deviation, the inter-spike interval is set and is random, unlike in Sequential mode where the interval is set at the beginning of the simulation.

3.7 APP_Out

The APP_Out receives the pulse from the Pulse_Generator (APP) at its input. When a pulse appears on one of the inputs (for Rev0, pulses will be synchronized with the clock), the APP_Out sub-module generates a unique AER packet, requests

the bus, waits until bus access is granted, and then sends an AER packet containing its unit address and address of the wire that received the pulse. In Random mode, any number of pulses out of 32 incoming bits from the Pulse_Generator sub-module can be asserted. In such a case, there can be more than one AER packet generated at a given clock cycle. These generated AER packets are stored in the FIFO (First In First Out). Furthermore, this sub-module keeps a log of the transactions generated.

3.8 APP_In

The APP_In watches all the bus transactions (i.e. AER packets being transmitted through the bus); when a valid transaction occurs, it checks to see if there is a match, via the Content Addressable Memory (CAM) [16] [17] [18] [19] [20]. The CAM structure contains four entries: the Valid (1 bit), the Source Unit (5 bit), the Source Wire (5 bit), and the Local Wire (5 bit) as shown in the figure 3.3. The CAM compares the least significant 10-bit field of the incoming AER packet with its entries (Source Unit-Source Wire pair). The Local Wire contains the index of the APP input wire on which the pulse in the target APP is to be generated, and an active pulse is generated if there is match (between least significant 10-bit field of the incoming AER packet and Source Unit-Source Wire pair in the CAM). The Valid bit is used to indicate whether or not a given location in the CAM is valid.

Content-addressable memory in APP_In module

Figure 3.3: Content Addressable Memory In The Rev0 DF Architecture

3.9 Bus_Control

The central component for the Rev0 DF architecture is the bus. It is a synchronous (clocked) bus that transmits the fixed 16-bit AER packets in 2 clock intervals. The arbitration is performed in the last clock of the interval. Bus arbitration is via a simple “daisy-chain.” Since buses in FPGAs are almost exclusively implemented using multiplexors as opposed to tri-state devices, a multiplexor-based bus architecture is assumed here. The Bus_Control is a test bench component; it can both send and receive the data on the bus.

3.9.1 Sub-Module Input/Output Signals:

3.9.1.1 Utility inputs to Bus_Control sub-module:

- Reset : active high, synchronous, resets all state in the Bus_Control sub-module to initial values.
- Clock : all inputs and outputs are synchronized by the rising edge of the clock.

3.9.1.2 Per bus access unit inputs to Bus_Control sub-module:

- Bus_xmt [7:0] : the packet data from the APP_In, one byte per clock cycle.
- Xmt_valid : output if asserted, indicates the least-significant byte (bits 7:0) of a two-byte AER packet valid on Bus_rcv [7:0]. The AER packet bits [15:8] follow on the next clock cycle.
- Bus_rqst : APP_In units request access to the bus by asserting Bus_rqst. Once asserted, Bus_rqst must remain asserted until Bus_grnt is asserted.

3.9.1.3 Per bus unit access unit Output from Bus_Control sub-module:

- Bus_valid : output if asserted indicates, the least-significant byte (bits 7:0) of a two-byte AER packet valid on Bus_rcv [7:0]. The AER packet bits [15:8] follow on the next clock cycle.
- Bus_rcv [7:0] : delivers AER packet data to APP_out one byte per clock cycle.

- Bus_grnt : output if asserted, bus access is granted to one of the requesting APP_Out sub-module. Once asserted, Bus_grnt will be active until the corresponding Bus_rqst input is de-asserted.

3.10 Rev0 DF Architecture Timing Diagram

The following diagrams show the timing of APP_In, APP_Out and arbitration bus sequences.

Figure 3.4: Rev0 DF Architecture APP_Out Timing Waveform

APP_In Single pulse bus sequence

Figure 3.5: Rev0 DF Architecture APP_In Timing Waveform

Bus arbitration between two APP_In units

Figure 3.6: Rev0 DF Architecture Bus Timing Waveform

3.11 Rev0 DF Architecture Summary

The Rev0 DF architecture consists of a set of bus access units and a single, synchronous, arbitrated, multi-access bus. The centralized shared bus arbitration scheme is easy to implement because the arbitration is relatively simple. In the arbitration scheme, the bus access units are designated with a priority level. The units with a higher priority get serviced first and those units are then granted bus access to transmit the AER packets. This bus arbitration scheme does have a disadvantage. The units with low priority may be locked out indefinitely, thus preventing their AER packets from being processed. In such scenarios, the AER packets either have to be dropped or stored in the buffer. As the AER packet generation rate is increased, the bus arbitration can act as a bottleneck leading to an increase in the AER packet drop rate or an increase in the buffer size requirement.

For experimental purposes, two bus access model simulators, arbitration and non-arbitration models, were implemented using Verilog. The simulation results are discussed in Chapter 7. In Conclusion, implementation of the Rev0 DF architecture is simple. However, the total number of AER packets the architecture can process during a fixed simulation time is limited because of the arbitration scheme. This leads to increased AER packet loss in the network. The minimal jitter noise (i.e. each wire sees the same delay for each spike it sends with random delays) from the source APP's output wires to the destination APP's input wires, the total AER packets delivered, and the total AER packets lost in the network determine the performance of the network. In this thesis, total AER packets delivered and the total number of AER packets lost is the evaluation metric used to determine the performance of the network. Hence, to address the issue of increased AER

packet loss, we move to the second iteration, 1-dimensional token ring network with separate APP address and connection spaces. The 1-dimensional token ring network is termed Rev1 DF architecture, and it is discussed in the next chapter.

Chapter 4

Rev1 DF Architecture

4.1 Rev1 DF Architecture Overview

The Rev1 DF architecture has nodes connected in a 1-dimensional token ring network [9] as shown in the figure 4.1. Each node consists of the following sub-modules: the APP, the AER/Convert, and the APP_Router. Further, to simplify the functionality of the APP_Router, 3 main sub-modules are defined in the APP_Router sub-module: the Router, the FIFO, and the APP_In as shown in figure 4.4. The nodes are interconnected with each other through channels formed from Ambric Registers (defined in the next section) as shown in the figure 4.3. For simulation purpose, 32 nodes are connected in the 1-dimensional token ring network fashion. Each node in the network are given an unique address (0 to 31) as shown in figure 4.4. The APP sub-module in each node is responsible for the pulse generation, and the AER/Convert sub-module is responsible for the AER packet generation.

Figure 4.1: Digital Fabric Architecture Rev1 System Diagram

4.2 Ambric Registers:

Figure 4.2: Ambric Registers: Interface between the nodes in Rev1

Ambric Registers are clocked registers with data in and data out. Along with data, there are two control signals, Valid and Accept. These signals implement a hardware protocol for forward and backward movement of data between the nodes. This protocol makes the systems using the Ambric Registers self-synchronizing and asynchronous. The protocol is simple, using only two signals and no intermediate logic is required. When a register is ready to accept data it asserts the Accept signal. It will assert the Valid signal when it has the output data available [21].

4.2.1 Pseudo-code for Ambric Registers:

Algorithm 1 module ambric_registers;

```
reg Accept, Valid;
reg [7:0] DataIn, DataOut;
always @(*)          // the accepter node process
begin
    Accept = 1'b1;    //the neighboring node is ready to accept the data.
    forever
        begin
            wait(Valid)
            DataIn = DataOut; // transfer the data
            Accept = 1'b0;
            wait(!Valid)
            Accept = 1'b1;
        end
    end
end
always @(*)          // the sender node process
begin
    Valid = 1'b0;
    forever
        begin
            wait(Accept)    // wait for neighboring node to accept the data.
            DataOut = $random; // generate random value and transfer the data.
            Valid = 1'b1;    // indicate the data is valid and ready to transfer
            wait(!Accept)
            Valid = 1'b0;
        end
    end
end
```

Each Ambric Register is preemptive, in that its operation is self-initiated. An upstream signal (Accept) is asserted when the register accepts an input and a downstream signal (Valid) is asserted when an output is available. The router sub-module in the APP_Router is responsible for controlling the Accept and Valid signals. The Accept signal is de-asserted (signal 0) by the router sub-module when the FIFO sub-module is full in its node. In such a scenario, the router sub-module does not accept any data (AER packets) from the neighboring nodes; instead, the router reads and processes the data from FIFO sub-module in its node, thus de-asserting the Accept signal. The AER packets, which are not accepted, are eventually dropped. Figure 4.3 shows the Rev1 DF Micro-architecture, connected in a token ring network fashion. The abbreviation, AR in the figure 4.3 represents the Ambric Register. Figure 4.4 shows the Rev1 APP_Router sub-module Micro-architecture and the way in which it interacts with the neighboring nodes.

4.3 Rev1 Micro-architecture System View One Dimension One Direction

Figure 4.3: Rev1 Micro-architecture System Diagram

4.4 APP_Router Sub-Module Micro-architecture

Figure 4.4: Rev1 APP_Router Sub-Module Micro-architecture

4.5 Clock domains in the Rev1 DF Architecture

Each node consists of the following sub-modules: the APP, the AER/Convert, and the APP_Router. Further, to simplify the functionality of the APP_Router, 3 main sub-modules are defined in the APP_Router sub-module: the Router, the FIFO, and the APP_In. In any node, the various sub-modules in the Rev1 DF architecture operate at different clock frequencies. Different clock frequencies were chosen because, during the random mode of operation, pulses appear randomly at random locations at the output of the APP sub-module, and multiple AER packets are generated in the same clock cycle. If the APP sub-module and the router sub-module (in the APP_Router sub-module) are operating at the same frequency, it becomes difficult to process all the AER packets, and there will be more AER packets dropped. In most designs, the needed multiple clock frequencies, are obtained by dividing a master clock (either synchronous division or asynchronous division), with help of clock frequency divider circuits (or Distributed Clock Manager in FPGAs). Here, for the Rev1 DF architecture, the master clock frequency is chosen to be 50 MHz (chosen because the on-board clock frequency available on Spartan 3E is 50 MHz). The subsequent lower clock frequencies are generated by dividing the master clock by a power of two (synchronous division). The other clock frequencies chosen for the operation of some of the sub-modules in the Rev1 DF architecture are 25 MHz and 6.25 MHz. The lower clock frequencies (25 MHz and 6.25 MHz) are multiple of the higher clock frequency (50 MHz) and thus, all the clock frequencies in the architecture are aligned (also reduce the problem of unpredictable behavior because of no proper synchronisation). The router sub-module within each APP_Router operates at the highest frequency (50 MHz) so

that it can process the AER packets faster while the APP sub-module in each node operates at the lowest frequency (6.25 MHz). Integrating the various modules or sub-modules, which are operating at different frequencies, can be troublesome because unpredictable behavior in the system could occur when there is no proper synchronization. The handshake technique is a very common method used for synchronisation across multiple clock domains. Here, this handshake technique is used between the APP (operating at 6.25 MHz) and the AER/Convert (operating at 25 MHz) sub-modules. When data are to be transferred across different clock domains, a FIFO is an ideal component. FIFOs in general help as a temporary storage buffer, which stores the data written from the write path until it is popped out by the receiver. Thus, in an application like a bridge across two protocol buses with different frequencies, FIFOs help in completing the bus cycles of a faster host sooner. Hence, for better performance of the system, a FIFO is used within the APP_Router sub-module to store the AER packets generated by the AER/Convert sub-module. Furthermore, the AER packets coming in from the neighboring nodes are not stored in this FIFO.

4.6 Address Event Representation Format: Rev1 DF Architecture

4.6.1 Different Modes Of Operation

- 000000 No-Op.
- 000001 Go-to mode: transmit spike to destination wire in destination unit.
- 000010 Broadcast mode: Broadcast spikes to destination wire in all units.

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 4.1: Address Event Representation Format In The Rev1 DF Architecture

For simulation purpose, 32 nodes are connected in the 1-dimensional token ring network fashion. The AER packet format for the Rev1 DF architecture is the same as the Rev0 DF architecture. Bits 15-10 in the AER packet determine the mode of operation, which are listed above and as shown in table 4.1. If the value is 000000, no operation is performed; if the value is 000001, the mode is set to Go-to; and if the value is 000010, Broadcast mode is set. In Broadcast mode, when any node routes the AER packets, it is assumed that every node adjacent to it (connected via the Ambric Registers), receives the AER packets, and eventually AER packets are broadcasted to all the nodes (i.e., there is just one sender, but the information is sent to all connected receivers). In Go-to mode, the AER packet is sent from a single source node to a specified destination node (i.e., there is just one sender, and one receiver). In Broadcast mode, the unit address in the AER packet (i.e., 9:6 in the AER packet, as shown in table 4.1) contains the source APP address (i.e., the source which generated the AER packet). While, in the Go-to mode, the unit address contains the destination APP address. The unit address and the wire address in the 16 bit AER packet, are formed with the help of a pseudo random number generator using a LFSR. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of the Modelsim simulation. In the Rev0 DF architecture, the Bus_Control sub-module decodes the information in the AER packet. In Rev1 and Rev2 DF architectures, the APP_Router sub-module is responsible for decoding

the AER packet information and performing the requested action. In the Go-to mode, the router sub-module is responsible for decoding the destination node information in the AER packet; calculate the shortest routing path in the network, and accordingly transmit the AER packet in the network. In Rev1 DF architecture, to realize the functionality of the Broadcast mode and transmit the AER packet to all the nodes in the network, certain rules are set in the router sub-module:

- Each node in the 1-dimensional token ring network (Rev1) is surrounded by 2 neighboring nodes (left and right) as shown in the figure 4.1. In Broadcast mode, any node generating the AER packet transmits the AER packet to 1 of its neighboring node out of the 2 nodes (no routing decision is done unlike Go-to mode). By default, the AER packet is transmitted via the Right channel to the node located to its right.
- In Broadcast mode, any node receiving the AER packet via the Right channel transmits the AER packet to its APP_In sub-module for data matching and to the node located to its right.
- Finally, broadcasting of an AER packet stops, when the AER packet reaches the node in the network which generated the AER packet.

The APP_Router sub-module in the node 0, is shown in the figure 4.4. Each APP_Router sub-module communicates with the neighboring APP_Router sub-modules (in the neighboring nodes) via Ambric Registers, which form a channels between nodes. The APP_Router sub-module in the node 0, accepts the data (Data_Route_In_Right_Channel) from the right APP_Router sub-module (31st APP_Router sub-module) and the data (Data_Route_In_Left_Channel) from the

left APP_Router sub-module (1st APP_Router sub-module) as shown in the figure 4.4.

The APP sub-module (Pulse_Generator) has two inputs, Clock and Reset, and one 32 bit output which drives the AER/Converter sub-module (Packet_Generator). The pulses are active high, one clock cycle long (6.25 MHz), and they can be generated in a variety of Sequential and Random patterns at regular intervals. The interval between pulse generations is a settable parameter. In the Sequential mode, only 1 bit out of every 32 bits are asserted, in sequence, at a given clock cycle. In the Random mode, the pulses appear randomly at a random location (in some scenarios all 32 bits can be asserted at regular intervals).

During Random pulse generation mode, more than one AER packet can be generated in the same clock cycle, and these packets have to be stored in order to be processed. The FIFO holds the to-be-processed packets generated from the AER/Convert sub-module in each node. The AER packets can go either to the left or to the right node depending on the shortest path decision made by the APP_Router sub-module processing the packet. If both the left and the right distances are equal, then the data is sent to the right node by default.

Since there are many AER packets being routed in the network at any given time, link contention is possible. This can occur, for example, when the AER packet from a neighboring node and the AER packet from a FIFO sub-module in the node want to take the same path in the network at the same time. In such a scenario, one of the AER packets is dropped depending on which packet has the higher priority. Certain assumptions are made in order to process the AER packets in such scenarios, which are listed in the next section. An increase in the number of spikes at the input of the AER/Converter sub-module will produce more AER

packets leading to increased link contention. Furthermore, as the time interval between the spikes decreases, the link contention also increases, and we can see the percentage increase in the packet drop, as discussed in Chapter 7.

4.7 Key assumptions in Rev1 DF architecture

This section summarizes various architecture and implementation assumptions:

- As there are many AER packets being routed in the network at a given time, there are chances of link contention, and so a decision has to be made as to which packet needs to be processed. Correction of the AER packet contention/link contention is not included in the simulation/FPGA prototyping for this thesis. Hence, the correct operation is assumed to be the acknowledgment of the AER packet delivery status (processed or dropped) to the node that initiated the process.
- When data are to be transferred across different clock domains, a FIFO is an ideal component. Here, the FIFO operates at the interface of 2 clock domains, a 25 MHz write frequency (Packet_Generator operating frequency) and a 50 MHz read frequency (Router sub-module operating frequency). In each node, the FIFO is used to store the AER packets generated by its node (generated by the Packet_Generator sub-module in the node). The AER packets arriving from the neighboring nodes are not stored in the FIFO.
- The APP_Router sub-module in association with the Ambric Registers determines whether the neighboring nodes are ready to accept and process each AER packet, if not, the AER packets (generated by its own node or arriving

from the neighboring node) are dropped. Each node accepts the AER packet from the neighboring nodes only if its FIFO sub-module is not full. Thus in such scenario, priority is given to the AER packets stored in the FIFO over the AER packets arriving from the neighboring nodes. The higher priority AER packet is transmitted, and the lower priority AER packet is dropped (since the AER packets coming in from neighboring nodes are not stored in FIFO). This type of AER packet loss is termed “AER packets lost due to FIFO full.”

- In any node, when the FIFO in the APP_Router sub-module is not full, it is possible for the AER packets arriving from the neighboring node and the AER packets generated from its own node (read from the FIFO sub-module) to want to take the same path in the network at the same time. In such a scenario, the AER packets arriving from the neighboring node have precedence. The higher priority AER packet is transmitted, and the lowest priority AER packet is dropped immediately. This type of AER packet loss is termed as “AER packets lost due to link contention.”
- Total AER packet loss in the network is the sum of the “AER packets lost due to FIFO full”, and the “AER packets lost due to link contention.” Simulations are performed, and results are tabulated in chapter 7 for different time intervals.
- The router sub-module in each APP_Router calculates the shortest delivery path for each AER packet. The APP_Router then transmits the AER packet to its destination in the network.

4.8 Rev1 DF Architecture Sub-Module Descriptions

Each node in the Rev1 DF Architecture consists of sub-modules namely: the APP (Pulse_Generator), the AER/Convert (Packet_Generator), and the APP_Router. Further, the APP_Router consists of following sub-modules: the Router, the FIFO and the APP_In.

4.9 Pulse_Generator or APP

Clock and reset form the input to the Pulse_Generator. The Pulse_Generator has a 32 bit output connected to the Packet_Generator. The Pulse_Generator (APP) communicates with the Packet_Generator (AER/Convert) via pulses, generated on its 32 bit output. Pulses are active high for one clock cycle. The Pulse_Generator can operate in either Random or Sequential mode at regular intervals. The interval between spikes is a settable parameter for both Sequential and Random mode. The interval is initially set at the beginning of simulation. In Sequential mode, only 1 bit out of 32 bits is asserted in sequence. In Random mode, random number of pulses appear randomly at random locations at the output of the Pulse_Generator sub-module. The Pulse_Generator operates at 6.25 MHz frequency.

4.10 Packet_Generator or AER/Convert

The Packet_Generator operates at 25 MHz. When a pulse appears on one of the lines, the Packet_Generator creates an AER packet. The AER packet format is as shown in table 4.1. The 16 bit AER packet consists of 3 segments namely: the operation mode (bits 15-10), the unit address (bits 9-6), and the wire address (bits 5-0). Bits 15-10 in the AER packet determine the mode of operation. The system

has 3 different modes of operation: No-Op, Go-to, and Broadcast mode. These modes are defined in detail in section 4.6. The unit address segment in an AER packet can be either the source APP address or the destination APP address (the source APP address is added to the AER packet if the mode is Broadcast; the destination APP address is added if the mode is Go-to) The wire address segment in an AER packet, is the address of the wire that received the pulse in the source APP. The unit address and the wire address in the 16 bit AER packet, are formed with the help of a pseudo random number generator using a LFSR. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of the Modelsim simulation. The AER Packets generated are stored in the FIFO.

4.11 APP_Router

4.11.1 Sub-Modules In APP_Router

The APP_Router consists of sub-modules namely: the Router, the FIFO, and the APP_In. These are described in the next section.

4.11.1.1 Router

The router sub-module in the APP_Router operates at 50 MHz frequency. The nodes in the Rev1 DF architecture are connected in 1-dimensional token ring network [12]. The nodes interact with each other through the channels formed by the Ambric Registers. The router sub-module interacts with the FIFO and the APP_In sub-modules to process the AER packets. In each node, the FIFO is used to store the AER packets generated by the AER/Convert sub-module. The router

sub-module steers the AER packets arriving from its FIFO or neighboring nodes, to the neighboring nodes. If the data in the AER packet matches the node address, then the AER packet is not routed and instead is sent to its APP_In sub-module for data matching. The 16 bit AER Packets (contains following information: the operation mode, the unit address and the wire address, as shown in table 4.1) are transmitted in 2 clock cycles (8 bits in each clock cycle). The router sub-module is responsible for making routing decisions based on the shortest path calculation (to send the AER packet to either the left or the right channel). It also makes a decision as to which AER packet (either an AER packet from it's FIFO or from a neighboring node) is to be processed or dropped during the link contention.

4.11.1.2 FIFO

When data are to be transferred across different clock domains, a FIFO is an ideal component. FIFOs in general help as a temporary storage buffer, which stores the data written from the write path until it is popped out by the consumer. The data values are written to the FIFO from one clock domain, and the data values are read from another clock domain. The FIFO is used to store the AER packets generated by the Packet_Generator (operating at 25 MHz). Hence, the FIFO operates at the interface of 2 clock domains, a 25 MHz write frequency (Packet_Generator operating frequency) and a 50 MHz read frequency (Router sub-module operating frequency). The read frequency depends on the APP_Router sub-module in the neighboring node. The write frequency depends on the number of AER packets generated each clock cycle (the output of the Packet_Generator sub-module), which, in turn, depends on how many pulses (output of the Pulse_Generator/APP)

are asserted (out of the 32 bits) at that particular clock cycle. For a FIFO capacity of 8 AER packets, the occupancy time of the AER packets in the FIFO will increase, depending on the following parameters: the operation mode, the interval between spikes. These can, in turn, increase the AER packet drop rate, as well. In any node, when the FIFO sub-module is full, the node does not accept any AER packets from the neighboring nodes, and subsequently the incoming AER packets arriving from the neighboring nodes are dropped.

4.11.1.3 APP_In

The APP_In watches all the bus transactions i.e., AER packets generated and transmitted by its node. When a valid transaction occurs, it checks to see if there is a match, via the Content Addressable Memory (CAM) [16] [17] [18] [19] [20]. The CAM structure contains four entries: the Valid (1 bit), the Source Unit (5 bit), the Source Wire (5 bit), and the Local Wire (5 bit) as shown in the figure 4.5. The CAM compares the least significant 10-bit field of the incoming AER packet with its entries (Source Unit-Source Wire pair). The Local Wire contains the index of the APP input wire on which the pulse in the target APP is to be generated, and an active pulse of one clock (50 MHz) duration is generated if there is match (between least significant 10-bit field of the incoming AER packet and Source Unit-Source Wire pair in the CAM). The Valid bit is used to indicate whether or not a given location in the CAM is valid.

Content-addressable memory in APP_In module

Figure 4.5: Content Addressable Memory In The Rev1 DF Architecture

4.12 Rev1 DF Architecture Summary

The arbiter scheme in the Rev0 DF architecture acts as the bottle neck that limits the performance (defined in section 3.11) of the network and the total number of packets processed/delivered is much less. Performance increases when moving from the Rev0 to the Rev1 DF architecture since the multiple links allow more concurrent packet transfer. The Rev1 DF simulator was implemented using Verilog. The results are discussed in chapter 7. Compared to the Rev0 DF architecture, the AER packet loss for the Rev1 DF architecture is much lower for the same packet

generation rate. Hence, the Rev1 DF architecture can handle the increased AER packet generation rate better than the Rev0 DF architecture.

Although the Rev1 DF architecture performs better in comparison to the Rev0 DF architecture, the Rev1 DF architecture sees a different type of packet loss not seen in the Rev0 DF architecture. This packet loss happens because of link contention. Link contention occurs when AER packets from multiple different sources in the network want to take the same path. The Rev1 DF architecture is connected in a 1-dimensional token ring, and the architecture does not provide enough channels for each individual node to connect to its neighboring nodes. This link contention leads to increased AER packets loss. To address the issue of increased AER packet loss due to link contention, for the next revision we moved to a 2-dimensional space where the nodes are connected in a 2-D torus. Also, the performance improvement seen in the Rev1 DF architecture compared to the Rev0 DF architecture gave us the confidence in the basic design to move from 1-dimension token ring network (Rev1) to a more complex 2D torus packet network (Rev2). Consequently, for the Rev2 DF architecture, the synchronous Rev1 DF architecture is expanded to a 2D torus packet network. The Rev2 DF architecture is discussed in the next chapter.

Chapter 5

Rev2 DF Architecture

5.1 Rev2 DF Architecture Overview

The Rev2 DF architecture is an extension of the Rev1 DF architecture. In the Rev2, the nodes are connected by a 2D torus packet network [9] as shown in the figure 5.1. The basic operation remains the same except that we extend the architecture into two dimensions, from 2 links per node to 4. Each node consists of the following sub-modules: the APP, the AER/Convert, and the APP_Router. The APP_Router further consists of the following sub-modules: the Router, the FIFO, and the APP_In. The nodes are interconnected with each other through channels formed by Ambric Registers (defined in the section 5.2) as shown in the figure 5.2. Each node has 4 surrounding nodes and hence four channels are used to communicate with the neighboring nodes (Left, Right, Up and Down). The APP sub-module in each node is responsible for pulse generation, and the AER/Convert sub-module is responsible for AER packet generation. Figure 5.2 represents the Rev2 DF architecture connected in a 2D torus network fashion. Each node addresses are composed of two parts : X and Y, which X shows the number of row and Y indicates the number of column that the node is located in figure 5.2. The letter's R, L, D and U in the figure indicate Left, Right, Down and

Up channels. Figure 5.2 represents a 3x3 2D torus network.

Figure 5.1: Digital Fabric Rev2 DF Architecture System Diagram

Figure 5.2: Digital Fabric Rev2 APP_Router Sub-Module Connectivity

5.2 Ambric Registers

Ambric Registers are clocked registers with data in and data out. Along with data, there are two control signals, Valid and Accept. When a register is ready to accept data it asserts the Accept signal. It will assert the Valid signal when it has the output data available [21]. The functionality of the Ambric Registers for the Rev2 DF architecture, remains same as the Rev1 DF architecture and is discussed in detail in section 4.2. The pseudo code for the Ambric Register is discussed in the section 4.2.1.

5.3 Rev2 APP_Router Sub-Module Micro-architecture

Figure 5.3: Rev2 APP_Router Sub-Module Micro-architecture

5.4 Clock domains in the Rev2 DF Architecture

The Rev2 DF architecture is an extension of the Rev1 DF architecture. In the Rev2, the nodes are connected by a 2D torus packet network [9]. The basic operation remains the same except that we extend the architecture into two dimensions, from 2 links per node to 4. Also, the Rev2 DF architecture consists of the same sub-modules as defined in the Rev1 DF architecture. Hence, in similar to the Rev1 DF architecture the various sub-modules in the Rev2 DF architecture operate at different frequencies and the handshaking protocols is the same (explained in detail in section 4.5).

5.5 Address Event Representation Format: Rev2 DF Architecture

5.5.1 Operation

- 000000 No-Op.
- 000001 Go-to mode: transmit spike to destination wire in destination unit.
- 000010 Broadcast mode: Broadcast spikes to destination wire in all units.

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 5.1: Address Event Representation Format In The Rev2 DF Architecture

For simulation purposes, nodes are connected in a 4x8 2D torus network (i.e., network has 4 rows and 8 columns). Each node has 4 surrounding nodes and hence four channels are used to communicate with its neighboring nodes (Left, Right,

Up and Down). The AER packet format for Rev2 is the same as the AER packet format defined for Rev0. Bits 15-10 in the AER packet determine the mode of operation, which are listed above and as shown in table 5.1. If the value is 000000, no operation is performed; if the value is 000001, the mode is set to Go-to; and if the value is 000010, Broadcast mode is set. In Broadcast mode, when any node routes the AER packets, it is assumed that every node adjacent to it (connected via the Ambric Registers), receives the AER packets, and eventually AER packets are broadcasted to all the nodes (i.e., there is just one sender, but the information is sent to all connected receivers). In Go-to mode, the AER packet is sent from a single source node to a specified destination node (i.e., there is just one sender, and one receiver). In Broadcast mode, the unit address in the AER packet (i.e., 9:6 in the AER packet, as shown in table 5.1) contains the source APP address (i.e., the source which generated the AER packet). While, in the Go-to mode, the unit address contains the destination APP address. The unit address and the wire address in the 16 bit AER packet, are formed with the help of a pseudo random number generator using a LFSR. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of the Modelsim simulation. In the Go-to mode, the router sub-module is responsible for decoding the destination node information in the AER packet; calculate the shortest routing path in the network, and accordingly transmit the AER packet. In Rev2 DF architecture, to realize the functionality of the Broadcast mode and transmit the AER packet to all the nodes in the network, certain rules are set in the router sub-module:

- In the Rev2 DF architecture, each node has 4 surrounding nodes as shown

in the figure 5.1, and hence four channels are used to communicate with the neighboring nodes (Left, Right, Up and Down). In Broadcast mode, any node generating the AER packet transmits the AER packet to 2 of its neighboring node out of 4 (no routing decision is done unlike Go-to mode). By default, the 2 neighboring nodes receiving the AER packets are the node located to its right and the node located below it (i.e., the nodes located to the right and below to the node which generated the AER packet).

- In Broadcast mode, any node receiving the AER packet via the Down channel, transmits the AER packet to its APP_In sub-module for data matching and to the node located below it (again via the Down channel).
- In Broadcast mode, any node receiving the AER packet via the Right channel transmits the AER packet to its APP_In sub-module for data matching and to the node located to its right and below it (via the Right and the Down channel respectively).
- Finally, broadcasting of an AER packet stops, when the AER packet reaches the row in the network where the node, which generated the AER packet, is located.

For example in the figure 5.2 if the node [1,1] generates an AER packet and the mode of operation is Broadcast, then the AER packet is transmitted to 2 of its neighboring nodes. In the figure, the letter's R, L, D and U indicate Left, Right, Down and Up channels.

- According to the rules set in the router sub-module, in this case the AER packet is transmitted to the node located to the right and to the node located

below $[1,1]$ (i.e., the node $[1,2]$ and the node $[2,1]$ receives the AER packet via the channel R4 and D4 respectively).

- The node $[1,2]$ upon receiving the AER packet from the node $[1,1]$ via the Right channel (R4 in the figure 5.2), transmits the AER packet to its APP_In sub-module for data matching. Also, the node $[1,2]$ transmits the AER packet to the node $[1,0]$ and to the node $[2,2]$ (via the Right channel-R5 and Down channel-D5 respectively).
- The node $[2,1]$ upon receiving the AER packet from node $[1,1]$ via the Down channel (D4 in the figure 5.2), transmits the AER packet to its APP_In sub-module for data matching and to the node located below it i.e., node $[0,1]$.
- The node $[2,2]$ upon receiving the AER packet from the node $[1,2]$ via the Down channel (D5 in the figure 5.2), transmits the AER packet to APP_In sub-module in its node and to the node $[0,2]$.
- Finally, broadcasting of an AER packet is stopped when it reaches any of the node located in the 1st row (i.e., either the node $[1,0]$ or $[1,1]$ or $[1,2]$).

For simulation purposes, nodes are connected in a 4×8 2D torus network. Each node consists of the following sub-modules: the APP, the AER/Convert, and the APP_Router. The APP_Router sub-module further consists of following sub-modules: the Router, the FIFO, and the APP_In. Each node communicates with its neighboring nodes via Ambric Registers, which form channels between the nodes. The APP_Router sub-module (for the node $[0,0]$ is shown in the figure 5.3.

The abbreviation AR stands for Ambric register. This node accepts AER packets from 4 different neighboring nodes (assuming 3x3 2D torus network):

- Data_Route_In_Right_Channel from the right node (node address [0,2])
- Data_Route_In_Left_Channel from the left node (node address [0,1])
- Data_Route_In_Up_Channel from the upper node (node address [1,0])
- Data_Route_In_Down_Channel from the down node (node address [2,0])

The signal names and the abbreviations used in the figure 5.3 are listed in Appendix A.

The APP sub-module (Pulse_Generator) has two inputs, Clock and Reset, and one 32 bit output driving the AER/Converter sub-module (Packet_Generator). Pulses are active high, one clock cycle long (6.25 MHz) and can be generated in a variety of Sequential and Random patterns at regular intervals. The interval between the pattern generation is a settable parameter. In the Sequential mode, only 1 bit out of 32 bits is asserted, in sequence. In Random mode, pulses appear randomly at random locations, at the output of Pulse_Generator sub-module (the worst-case scenario would be that all 32 bits are asserted at regular intervals). During Random mode, more than one AER packet can be generated in a single clock cycle, and these AER packets have to be stored in order to be processed. Thus, the FIFO holds the AER packets generated from the AER/Convert sub-module in each node. The AER packet generated can go to either to the Left, Right, Down, or Up nodes depending on the shortest path decision made by the node processing the packet.

5.6 Assumptions

This section summarizes the various architecture and implementation assumptions:

- As there are many AER packets being routed in the network at a given time, there are chances of link contention, and so a decision has to be made as to which packet needs to be processed. Correction of the AER packet contention/link contention is not included in the simulation/FPGA prototyping for this thesis. Hence, the correct operation is assumed to be the acknowledgment of the AER packet delivery status (processed or dropped) to the node that initiated the process.
- When data are to be transferred across different clock domains, a FIFO is an ideal component. Here, the FIFO operates at the interface of 2 clock domains, a 25 MHz write frequency (Packet_Generator operating frequency) and a 50 MHz read frequency (Router sub-module operating frequency). In each node, the FIFO is used to store the AER packets generated by its node (generated by the Packet_Generator sub-module in the node). The AER packets arriving from the neighboring nodes are not stored in the FIFO.
- The APP_Router sub-module in association with the Ambric Registers determines whether the neighboring nodes are ready to accept and process the AER packets, if the nodes are not ready, the AER packets are dropped. Each node accepts an AER packet from the neighboring node only if its FIFO sub-module is not full. Once the FIFO is full, priority is given to the AER packets stored in the FIFO over the AER packets arriving from the neighboring nodes. The higher priority AER packet is transmitted, and the

lower priority AER packet is dropped (since the AER packets coming in from neighboring nodes are not stored in FIFO). This type of an AER packet loss is termed as “AER packet lost due to FIFO full.”

- In any node, when the FIFO in the APP_Router sub-module is not full, it is possible for the AER packets arriving from the neighboring node and the AER packets generated from it’s own node (read from the FIFO sub-module) to want to take the same path in the network at the same time. In such a scenario, the AER packets arriving from the neighboring node have priority. Also, in certain scenarios AER packets arriving from different neighboring nodes, might want to take the same path in the network at the same time. Hence, priority rules are set by the router sub-module, in order to handle the AER packets in such scenarios. The AER packet arriving from the Left channel has the highest priority, followed by the Right channel, the Down channel and then the Up channel. The higher priority AER packet is transmitted, and the lower priority AER packet is stored in a buffer of size 1. The lower priority AER packet sits in the buffer for 6 simulation clock cycle, and the router sub-module keeps checking if the desired channel is free to process this low priority AER packet. Later, after 6 simulation clock cycles, if the channel is still not free or if it is replaced by another AER packet, then the AER packet waiting in the buffer is dropped immediately. This type of AER packet loss is termed as “AER packet lost due to link contention.”
- Total AER packet loss in the network is the sum of the “AER packets lost due to FIFO full”, and the “AER packets lost due to link contention.” Simulations are performed, and results are tabulated in chapter 7 for different

time intervals.

- The router sub-module in each APP_Router sub-module calculates the shortest delivery path for each AER packet. The APP_Router sub-module then transmits the packet to its destination in the network.

5.7 Rev2 DF Architecture Sub-Module Descriptions

Each node in the Rev2 DF Architecture consists of sub-modules namely : the APP (Pulse_Generator), the AER/Convert (Packet_Generator), and the APP_Router. Further, the APP_Router consists of following sub-modules: the Router, the FIFO and the APP_In.

5.8 Pulse_Generator or APP

Clock and reset form the input to the Pulse_Generator sub-module. The Pulse_Generator has 32 bit output connected to the Packet_Generator. The Pulse_Generator (APP) communicates with the Packet_Generator (AER/Convert) sub-module via pulse over 32 bit output. Pulses are active high for one clock cycle. The Pulse_Generator can operate in either Random or Sequential mode at regular intervals. The interval between spikes is a settable parameter for both Sequential and Random mode. The interval is initially set at the beginning of simulation. In Sequential mode, only 1 bit out of 32 bits is asserted, in sequence. In Random mode, random number of pulses appear randomly at random locations at the output of the Pulse_Generator sub-module. The Pulse_Generator operates at 6.25 MHz frequency.

5.9 Packet_Generator or AER/Convert

The Packet_Generator operates at 25 MHz. When a pulse appears on one of the lines, the Packet_Generator creates an AER packet. The AER packet format is as shown in table 5.1. The 16 bit AER packet consists of 3 segments namely: the operation mode (bits 15-10), the unit address (bits 9-6), and the wire address (bits 5-0). Bits 15-10 in the AER packet determine the mode of operation. The system has 3 different modes of operation: No-Op, Go-to, and Broadcast mode. These modes are defined in detail in section 5.5. The unit address segment in an AER packet can be either the source APP address or the destination APP address (the source APP address is added to the packet if the mode is Broadcast; the destination APP address is added if the mode is Go-to) The wire address segment in an AER packet, is the address of the wire that received the pulse in the source APP (this forms the data that is matched against the Content Addressable Memory in APP_In sub-module). The unit address and the wire address in the 16 bit AER packet, are formed with the help of a pseudo random number generator using a LFSR. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of the Modelsim simulation. The AER Packets generated are stored in the FIFO.

5.10 APP_Router

5.10.1 Sub-Modules In APP_Router

The APP_Router has sub-modules named Router, FIFO and APP_In, and these sub-modules are described in the next section.

5.10.1.1 Router

The router sub-module in the APP_Router operates at 50 MHz frequency. The nodes in the Rev1 DF architecture are connected in 2D torus network fashion. The nodes interact with each other through the channels formed by the Ambric Registers. The router sub-module interacts with the FIFO and the APP_In sub-modules to process the AER packets. The FIFO is used to store the AER packets generated by the AER/Convert sub-module. The router sub-module steers the AER packets arriving from its FIFO or neighboring nodes, to the neighboring nodes. If the data in the AER packet matches the node address, then the AER packet is not routed and instead is sent to its APP_In sub-module for data matching. The 16 bit AER packets (contains following information: the operation mode, the unit address and the wire address, as shown in table 4.1) are transmitted in 2 clock cycles (8 bits in each clock cycle). The router sub-module is responsible for making routing decisions based on the shortest path calculation (to send the AER packet to either the Left, Right, Up or Down channel). It also makes a decision as to which AER packet (either an AER packet from it's FIFO or from a neighboring node) is to be processed or dropped during the link contention.

5.10.1.2 FIFO

When data are to be transferred across different clock domains, a FIFO is an ideal component. FIFOs in general help as a temporary storage buffer, which stores the data written from the write path until it is popped out by the receiver. The data values are written to the FIFO from one clock domain, and the data values are

read from another clock domain. The FIFO is used to store the AER packets generated by the Packet_Generator (operating at 25 MHz). Here, the FIFO operates at the interface of 2 clock domains, a 25 MHz write frequency (Packet_Generator operating frequency) and a 50 MHz read frequency (Router sub-module operating frequency). The read frequency depends on the APP_Router sub-module in the neighboring node. The write frequency depends on the number of AER packets generated each clock cycle (the output of the Packet_Generator sub-module), which, in turn, depends on how many pulses (output of the Pulse_Generator/APP) are asserted (out of the 32 bits) at that particular clock cycle. For a FIFO capacity of 8 AER packets, the occupancy time of the AER packets in the FIFO will increase, depending on the following parameters: the operation mode, the interval between spikes. These can, in turn, increase the AER packet drop rate, as well. In any node, when the FIFO sub-module is full, the node does not accept any AER packets from the neighboring node, and subsequently the incoming AER packets arriving from the neighboring node are dropped.

5.10.1.3 APP_In

The APP_In watches all the bus transactions i.e., AER packets generated and transmitted by its node. When a valid transaction occurs, it checks to see if there is a match, via the Content Addressable Memory (CAM) [16] [17] [18] [19] [20]. The CAM structure is same as defined in the Rev1 DF architecture and is shown in the figure 4.5. The CAM contains four entries: the Valid (1 bit), the Source Unit (5 bit), the Source Wire (5 bit), and the Local Wire (5 bit) as shown in the figure 4.5. The CAM compares the least significant 10-bit field of the incoming AER packet with its entries (Source Unit-Source Wire pair). The Local Wire

contains the index of the APP input wire on which the pulse in the target APP is to be generated, and an active pulse of one clock duration (50 MHz) is generated if there is match (between least significant 10-bit field of the incoming AER packet and Source Unit-Source Wire pair in the CAM). The Valid bit is used to indicate whether or not a given location in the CAM is valid.

Chapter 6

Modelsim And FPGA

6.1 Chapter Overview

This chapter provides an overview of the implementation and the instrumentation techniques used to realize the above defined architectures namely, Rev0, Rev1 and Rev2 DF architectures. The types of tools and their versions, the FPGA device and its specifications, its purpose, and how it is utilized are also described. Modelsim [23] is used for Verilog simulation. The Spartan 3E student kit FPGA [14], along with Xilinx ISE tool, is used for FPGA programming. This chapter also discusses the Microblaze RISC Processor, which is used to control the LCD display, Rs232 communications, and the VGA display in the Spartan 3E FPGA board.

6.2 Modelsim

A simulation of the Verilog is performed using Modelsim (Modelsim PE Student Edition 6.5d). Modelsim has the necessary performance capabilities and capacity to simulate larger blocks and systems. In this work, Verilog is used to implement the architectures. Simulations performed with the help of Modelsim help us

demonstrate the correct operation of the Rev0, Rev1, and Rev2 DF architectures. For Rev1 DF architecture, the next step of the process is performed where the design is synthesized and ported to an FPGA.

6.3 Approach in implementing Rev0 DF Architecture

The Rev0, Rev1, and Rev2 DF architectures are implemented using Verilog. The Rev0 DF architecture has a single, synchronous, shared bus. There are two kinds of bus access units, APP_In and APP_Out. The APP_Out receives a pulse from the APP on dedicated lines, and these are translated into AER packets and then broadcast on the bus. The APP_In receives the data from the bus in the form of AER packets, which are decoded and converted to pulses. These pulses are transmitted as input to the addressed lines on the APP. The APP_In has an internal memory (CAM), that scans the bus for any match between an address on the bus and an address in the CAM. If a match is found, the CAM indicates on which APP input wire the pulse has to be generated.

For implementation purposes in Verilog, the Rev0 DF architecture is divided into 4 main sub-modules: the APP_In, the APP_Out, the Pulse_Generator (acts as APP) and the Bus_Control. The Pulse_Generator sub-module can operate in either Random or Sequential mode at regular intervals. The interval between generated spikes is a settable parameter. In Sequential mode, only 1 bit out of 1023 is asserted at a time. In Random mode, the appearance of a pulse at the output is random. The functionality of generating random pulses at the output is achieved in Verilog with the help of the probabilistic distribution function: \$Dist_normal function (seed, mean, and standard deviation are the arguments of the function). This

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 6.1: Address Event Representation Format For Rev0, Rev1, and Rev2 DF Architecture

built-in function in Verilog, is a random number generator that returns integer values distributed according to standard probabilistic functions. This function presents a way to test a design using randomly generated data. The \$Dist_normal function is not synthesizable.

Upon the arrival of a pulse at the input of the APP_Out sub-module, the pulse is converted into an AER packet and is stored in the FIFO. The format of the generated AER packet remains same for Rev0, Rev1, and Rev2 DF architecture. The AER packet format is shown in the above table and is now discussed.

6.4 Address Event Representation Packet Format

The Address Event Representation packet format is shown in the above table 6.1. The operation mode can be configured to No-Op or to Go-to or to broadcast mode in the APP_Out sub-module. If the operation is in No-Op mode, then the unit address and the wire address are filled with “0”. If the operation is in Go-to mode the unit address is filled with a destination address. The destination address is generated in the Packet_Generator with the help of the pseudo random number generator, achieved in Verilog with the help of LFSR (defined in section 2.5). The sequence generated is not exactly random because it repeats eventually. However, for the purpose of evaluating system performance, it can be considered random. If the operation is in broadcast mode, the unit address is filled with the source APP address that generated the AER packet. The wire address generation for both the

modes remains the same, and the wire address is filled with a value that represents the input line that is asserted. For example, if the 10th input (out of 32 inputs) is asserted, then wire address is filled with 10 (binary value). The top module instantiates all the components and acts as a test bench. The registers defined in the top module store the statistics such as the “Total Packets Generated”, the “Total Packets Lost”, and the “Occupancy Time Of Data In The FIFO” (these statistics are detailed in section 7.2). These statistics are written into a file at the end of the simulation. The pulse generation rate can be controlled in the Pulse_Generator sub-module by varying the arguments in \$Dist_normal function: the mean and the standard deviation. The statistics are calculated for different simulations performed at varying time intervals, and these statistics are tabulated in the result’s section in Chapter 7.

6.5 Approach in implementing Rev1 DF Architecture

The Rev1 DF architecture was implemented again using Verilog, and is simulated and verified using Modelsim. It is also synthesized and ported to the FPGA Board. The Rev1 DF architecture is a 1-dimensional token ring network. To simulate the architecture in Verilog, three main sub-modules are defined, the Pulse_Generator (APP), the Packet_Generator (AER/Packet Converter), the APP_Router. Further, the APP_Router consists of following sub-modules: the Router, the FIFO and the APP_In. The Packet_Generator functionality is the same as it was for the Rev0 DF architecture. Furthermore, the operation mode (either Go-to or broadcast mode) can be set in the Packet_Generator. Since the Rev1 DF architecture is synthesized on the FPGA board, \$Dist_normal (seed, mean, standard deviation)

cannot be used for Pulse generation in the Pulse_Generator sub-module as this function is not synthesizable. Hence, for Rev1, in order to generate random pulses at the output of the Pulse_Generator (APP), a LFSR is used. The top module instantiates all these components, and also serves as a test bench. Similar to Rev0, registers are defined in the top module for the Rev1 DF architecture, and they store the statistics (defined in section 7.2) such as the “Total Packets Generated, the “Total Packets Dropped”, and the “Occupancy Time Of Data In The FIFO”, after the execution (stored for both simulation and FPGA prototyping).

6.6 Approach in Implementing Rev2 DF Architecture

The Rev2 DF architecture is an extension of the Rev1 DF architecture (1-dimension token ring network) to a 2D torus packet network. The implementation methodology remains the same, except that router functionality is increased to allow for vertical routing paths and more complex routing.

6.7 Xilinx Spartan 3E FPGA Board

The FPGA board used for demonstration purposes is the Xilinx Spartan 3E Starter Kit[14]. The board provides a powerful, advanced, self-contained development platform. It features 500k gates with an embedded RISC processor.

The Xilinx embedded tool chain consists of three major GUI-oriented applications and many smaller applications that are invoked by the major applications. The three major tools in the Xilinx embedded tool chain are:

- ISE: This tool enables designers to synthesize their designs, place and route; do a timing analysis; simulate the design for different stimulus, and configure

the target device.

- XPS: This tool enables designers to configure and build the hardware specification of their embedded system (processor core, memory-controller, I/O peripherals, etc.). XPS converts the designer's platform specification into synthesizable RTL. Its GUI provides a point and click interface to create Microblaze based systems. The Microblaze, is a 32 bit RISC architecture soft processor core, with a instruction set optimized for embedded applications. For the Microblaze core, the XPS generates an encrypted netlist.
- SDK: This tool is a software development kit based on the Eclipse platform and the GNU tool chain. The ISE, XPS and SDK, work together to provide an integrated environment for building the SOC hardware and software for Xilinx FPGAs.

6.8 Microblaze RISC processor

The board is also compatible with the Microblaze Embedded Development Kit. Microblaze [22] is a synthesizable 32- bit RISC soft processor core that implements a wide range of system tasks. The basic architecture consists of 32 general-purpose registers, an ALU, a shift unit and two levels of interrupt.

Figure 6.1: Microblaze Processor : RISC 32 bit Soft Core Processor from Xilinx

6.9 Implementation on the FPGA Board Using Microblaze

For the work presented here, Microblaze is used to control the LCD display, the RS232 communications, and the VGA display. During simulation of the Rev1 DF architecture, various statistics (defined in section 7.2) such as total AER packets generated, total AER packets dropped, and occupancy time of the system are calculated in the top module. These statistics are stored in registers, and then sent over the General Purpose Input/Output (GPIO) channel to the Microblaze processor. The Microblaze can be controlled using C/C++, and consequently, it

is easier to send the results to the LCD display or the RS232 port. With the help of RS232 port, results can then be displayed on the terminal of the computer. In the next section, details of the GPIO channel, the UART module, and the LCD display are explained.

Figure 6.2: Rev1 DF Architecture on FPGA

6.10 GPIO: General Purpose Input/Output

The XPS GPIO is a 32-bit peripheral that attaches to the Processor Local Bus (PLB). The XPS GPIO design provides a general-purpose input/output interface to a PLB. The XPS GPIO can be configured as either a single or dual channel device. The channel width is configurable, and when both channels are enabled,

the channel width of each of the channels can be configured individually. The main interface and modules of GPIO design are:

- PLB interface module
- Interrupt control
- GPIO core

Figure 6.3: GPIO Software IP by Xilinx

6.10.1 PLB Interface Module

The PLB Interface module provides interface between the GPIO core and the PLB bus standard. The PLB Interface module implements the basic functionality of the PLB slave operation.

6.10.2 Interrupt Control

The interrupt controller provides interrupt capture support for the GPIO core. The interrupt controller is used to collect the interrupts from the GPIO core, by which the GPIO core requests the attention of the microprocessor through the assertion of interrupt signals.

6.10.3 GPIO Core

The GPIO core provides an interface between the Intellectual Property Inter Connectivity (IPIC) interface and the XPS GPIO channels. The GPIO core consists of registers and multiplexers for reading and writing the XPS GPIO channel registers.

6.11 UART: Universal Asynchronous Receiver Transmitter

The XPS Universal Asynchronous Receiver Transmitter (UART) Lite Interface connects to the PLB (Processor Local Bus) and provides the controller interface for asynchronous serial data transfer.

6.11.1 Functional Description

The XPS UART Lite performs parallel-to-serial conversions on characters received through PLB and serial to parallel conversion on characters received from a serial peripheral. The XPS UART Lite is capable of transmitting and receiving 8, 7, 6 or 5 bit characters, with 1 stop bit and odd, even or no parity. The XPS UART Lite can transmit and receive independently. The device can be configured, and its status can be monitored via the internal register set. The XPS UART Lite generates an interrupt when data is present in the Receive FIFO or when

the Transmit FIFO becomes empty. This interrupt can be masked by using an interrupt enable/disable signal. The device contains a 16-bit programmable baud rate generator, and an independent 16 word Transmit and Receive FIFO.

Chapter 7

Results

7.1 Chapter Overview

In this chapter execution results for the Rev0, Rev1 and Rev2 DF architectures are presented. The simulations involved collecting statistics (described in section 7.2) based on different settings of the FIFO queue and for different pulse generation rates. The Rev1 DF architecture is synthesized for an FPGA, and the results are displayed on the LCD and the Hyper-Terminal.

7.2 Statistics Definitions

Some of the statistics collected, at the end of the simulation for Rev0, Rev1 and Rev2 are:

- **Total Packets Generated** : When a pulse appears on one of the lines of the AER/Convert sub-module in each node, it creates an AER packet. The total number of AER packets generated is the sum of all the AER packets generated by 32 AER/Convert sub-modules in the network at the end of the simulation.
- **Total Packets Lost** : The sum of the AER packets lost due to FIFO full, and

the packets lost due to link contention by all 32 nodes in the network. The total packet loss for the Rev0 DF architecture is just AER packets loss due to FIFO full, since there is no link contention in the arbitration scheme.

- **Occupancy Time Of Data In The FIFO :** This is the percentage of time for which the AER packet sits in a FIFO against being routed by APP_Router sub-module in each node.

7.3 Modelsim Simulation Results

Simulations of the various architectures described above are performed using the Modelsim environment (Modelsim PE Student Edition 6.5d). In the next sections, some of the results for Rev0, Rev1 and Rev2 DF architectures are discussed. From the simulation, the above mentioned statistics are calculated. In the next sections, statistics such as the total AER packets generated and the total AER packets lost in the network for different configurations are discussed and their results are plotted. For all the results discussed below, X-axis represents “Time in microseconds” (in steps of 50 microseconds or 10 microseconds depending on the packet generation rate) and Y-axis represent “Total Packets Generated”. Here, the total AER packets lost during a fixed simulation time is the evaluation metric to determine the performance of a network (Rev0, Rev1 and Rev2).

7.4 Rev0 DF Architecture Results: Operation Mode: Go-to

The results for the Rev0 DF architecture operating in the Go-to mode are discussed in this section. Figure 7.1, shows a comparison between the arbitration and the non-arbitration model.

Figure 7.1: Rev0 Simulation Results Arbitration vs. Non-Arbitration Model

In the arbitration model, all APP_Out units send a request. Those APP_Out which are not granted access to the bus keep trying with a particular packet until access is granted. In the non-arbitrated model, the multiplexor structure guarantees that only one unit, which has the highest priority, accesses the bus in the arbitration cycle. However, all requests assume their packets were delivered. So AER packets from lower priority units are lost. From figure 7.1, we can see AER packet's loss for non-arbitration model is more in comparison to the arbitration model.

7.4.1 Rev0 DF Architecture Simulation results: Varying Packet Generation Rate

Figure 7.2: Rev0 Simulation Results Go-to Mode : Lower Packet Generation Rate

Figures 7.2 and 7.3 show statistics obtained for arbitration model for different AER packet generation rates. Figure 7.2 represents statistics obtained with lower AER packet generation rates, while figure 7.3 represents statistics obtained with higher AER packet generation rates (in the figure 7.3, the number of AER packets generated is around to 15k while in the figure 7.2 AER packets generated is around to 2k per 100 microseconds).

Figure 7.3: Rev0 Simulation Results Go-to Mode : Higher Packet Generation Rate

The term efficiency is defined as a percentage of total AER packets lost to total AER packets generated at a fixed simulation time. From figure 7.2, we can see that the Rev0 DF architecture has an efficiency closer to 87% (at 30 microseconds in the figure 7.2) when the AER packet generation rate is very low. As, we increase the packet generation rate (figure 7.3), the efficiency falls as low as 12% (at 100 microseconds in the figure 7.3). Also, from the figure 7.2 we can see that at lower AER packet generation rate, FIFO size of 16 gives improved performance in comparison to FIFO size of 8. As, we increase the AER packet generation rate (figure 7.3), increasing the FIFO size does not help much in reducing the AER packet loss, and the FIFO size requirement to obtain better performance

also increases.

In conclusion, we can see that Rev0 DF architecture has better performance when AER packets are generated at a lower rate. As, we dial up the packet generation rate the network fails to handle the packets, and the performance of the network deteriorates. There is not much performance improvement when the FIFO size is increased.

7.5 Rev1 DF Architecture Simulation Results

7.5.1 Address Event Representation Format: Rev1 DF Architecture

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 7.1: Revisiting Address Event Representation Format In The Rev1 DF Architecture

Bits 15-10 in the AER packet determine the mode of operation as shown in the above table 7.1. If the value is 000000, no operation is performed. If the value is 000001, the mode is set to Go-to mode, and if the value is 000010, Broadcast mode is set. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of Modelsim simulation. The other information in the AER packet, the unit address and the wire address, are formed with the help of a pseudo random number generator using a LFSR.

7.5.2 Rev1 DF Architecture Results: Operation Mode: Go-to

Some of the statistics calculated during the simulations are plotted and shown below, operating in Go-to mode with random pulse generation. The statistics obtained are cumulative through time (in microseconds). The network statistics, like total AER packets generated and total AER packets lost vs. time in microseconds, are shown in the below figures. Figure 7.5 and 7.4 show network statistics for different pulse generation rates. By varying the pulse generation rate (controlling the output of the APP sub-module), we can vary packet generation at a given clock cycle.

7.5.3 Rev1 DF Architecture Simulation Results: Varying The Packet Generation Rate

Figure 7.4: Rev1 Simulation Results Go-to Mode: Lower Packet Generation Rate

The total AER packets lost in the Rev1 DF architecture is the sum of the “AER packets lost due to FIFO full”, and “AER packets lost due to link contention” (defined in section 4.7). Figures 7.4 and 7.5 show statistics for lower and higher packet generation rates respectively. With the help of figure 7.4 we can calculate the efficiency of the network to be 78%. As, we increase the AER packet generation rate the efficiency of the network falls as low as 55%.

Figure 7.5: Rev1 Simulation Results Go-to Mode: Higher Packet Generation Rate

The main factor, which, affects the total AER packet loss is the “AER packets lost due to link contention”, where in 2 different packets arriving from a different source who want to take the same path in the network. Hence, from the figure 7.5 and 7.4 we can see that varying the FIFO size does not help much in reducing the total AER packet loss. Since the APP sub-module and AER/Convert sub-module in each node are operating at a lower frequency (6.25 MHz and 25 MHz respectively) compared to the router sub-module (50 MHz) in each APP_Router, packet loss due to FIFO being full is less compared to packet loss due to link contention.

7.5.4 Rev0 and Rev1 DF Architectures Result Comparison

Figure 7.6: Comparison between Rev0 and Rev1 DF Architecture

The Rev0 DF architecture consists of a set of bus access units and a single, synchronous, arbitrated, multi-access bus and the Rev1 DF architecture has nodes connected in a 1-dimensional token ring network [9]. Figure 7.6 shows a comparison between Rev0 and Rev1 performance. From the figure, we can see that, for the same packet generation, the total AER packets lost in the Rev1 DF architecture is much less than for the Rev0 DF architecture. The performance of the Rev0 DF architectures degrades when we increase the AER packet generation rate. The efficiency of the architectures were 11% and 67% respectively for the Rev0 and Rev1

DF architectures at very high packet generation rates. The Rev1 DF architecture is not based on a centralized shared arbitration scheme as in the Rev0 DF architecture which proved to be the bottleneck (single bus vs. multiple links) to better performance. Performance increases when moving from the Rev0 to the Rev1 DF architecture since the multiple links allow more concurrent packet transfer. The Rev1 DF architecture provides separate APP address and connection spaces and is connected in 1-dimension token ring fashion. Because of the token ring, the AER packets arriving from multiple sources have 2 channels (left or right) to traverse between the nodes in the network. Hence, when the AER packet generation rate is increased, many packets from different sources take the same path in the network, which leads to increased AER packet loss due to link contention.

7.6 Rev2 DF Architecture Simulation Results

In the Rev2 DF architecture, we have each node connected in a 2D torus. For simulation purposes, a 4x8 2D torus network is simulated. The Rev2 DF architecture is implemented in Verilog HDL. Again, the Modelsim PE Student Edition 6.5d is used for simulation.

7.7 Address Event Representation Format: Rev2 DF Architecture

Bits 15-10 in the AER packet determine the mode of operation as shown in the table 7.2. If the value is 000000, no operation is performed. If the value is 000001, the mode is set to Go-to mode, and if the value is 000010, Broadcast mode is set. The mode of operation (i.e., the first 6 bits of the AER packet) can be set to either Go-to or Broadcast at the beginning of Modelsim simulation. The other

15....10	9....6	4....0
Operation	Unit Address	Wire Address

Table 7.2: Revisiting Address Event Representation Format In The Rev2 DF Architecture

information in the AER packet, the unit address and the wire address are formed with the help of a pseudo random number generator using a LFSR.

7.7.1 Rev2 DF Architecture Results: Operation Mode: Go-to

Some of the statistics calculated during the simulations are plotted and shown below operating in Go-to mode with random pulse generation. The statistics obtained are cumulative through time (in microseconds). The network statistics, such as the total number of AER packets generated, and the total number of AER packets lost vs. the time in microseconds are shown in figures. Figure 7.8 and 7.7 show network statistics for different pulse generation rates. By varying the pulse generation rate (controlling the output of APP sub-module), we can vary packet generation at a given clock cycle. Figure 7.8 shows statistics for higher pulse generation rates, hence generating more AER packets at a given clock cycle.

7.7.2 Rev2 DF Architecture Simulation results: Varying Packet Generation Rate

Figure 7.7: Rev2 Simulation Results Go-to Mode: Lower Packet Generation Rate

The total number of AER packets lost in the Rev2 DF architecture is the sum of all the “AER packets lost due to FIFO full”, and the “AER packets lost due to link contention” (defined in section 5.6). Figures 7.7 and 7.8 show statistics for lower and higher packet generation rates respectively. From figures 7.7 and 7.8 it is seen that the efficiency for lower packet generation is around 98% and for higher packet generation it is 96%. So, the percent decrease in the efficiency of the network as we increase the AER packet generation rate is around 70%, 22% and

3% for Rev0, Rev1 and Rev2 DF architectures respectively. Hence, the Rev2 DF architecture, performs the best and can handle increased AER packet generation rate effectively, compared to Rev0 and Rev1 DF architecture.

Figure 7.8: Rev2 Simulation Results Go-to Mode: Higher Packet Generation Rate

Also from the figure 7.8 and 7.7 we can see that varying the FIFO size does not help much in reducing the packet loss because most packet loss in the network is due to link contention (i.e., a packet from a AER/Convert sub-module in any node and a packet from the neighboring node wants to take the same path in the network during the same clock cycle). In each node, since the APP sub-module and the AER/Convert sub-module are operating at a lower frequency (6.25 MHz and 25 MHz respectively) compared to the router sub-module (50 MHz), the “AER

packet loss due to FIFO full”, is lower than the “AER packet loss due to link contention.”

7.7.3 Rev0, Rev1 and Rev2 DF Architectures Results Comparison

Figure 7.9: Rev0, Rev1 and Rev2 DF Architectures Comparison plot

The total AER packets lost in the network is the sum of the “AER packets lost due to FIFO full”, and the “AER packets lost due to link contention”. The efficiency is defined as a percentage of the total number of AER packets lost to the total number of AER packets generated during a fixed simulation time. Efficiency is used as the evaluation metric to define the performance of the above discussed architectures. Figure 7.9 shows a comparison among Rev0, Rev1 and Rev2. From

the figure, we can see that for the same packet generation rate, the total AER packets lost in the Rev2 DF architecture is much lower compared to packet loss rate in the Rev0 and Rev1 DF architectures. The statistics shown in figure 7.9 are for a FIFO size of 8. The Rev0 DF architecture does not support high AER packet routing. Because the Rev0 DF architecture uses a multiplexor based bus, it does not experience link contention, but AER packets are dropped due to the FIFO being full. The arbiter scheme in Rev0 acts as a bottle neck, and the performance of the architecture deteriorates as we increase the AER packet generation rate. For the Rev1 DF architecture, we implemented a 1-dimension token ring network. For the Rev1 DF architecture, the number of AER packets lost due to the FIFO being full was reduced in comparison to Rev0 DF architecture, but there is still a problem of link contention, which leads to AER packet loss as we increase the AER packet generation rate.

The Rev2 DF architecture, performs the best compared to the Rev0 and Rev1 DF architectures. Compared to Rev0 and Rev1, the Rev2 DF architecture has much lower AER packet loss due to the FIFO being full. Also, in Rev2 DF architecture, the AER packet loss due to link contention was reduced compared to the Rev1 DF architecture. Thus, the total AER packet lost for Rev2 DF architecture is much lower compared to the Rev0 and Rev1 DF architectures and is shown in figure 7.9. Hence, from figure 7.9 we can conclude that the Rev2 DF architecture is more efficient compared to the Rev0 and Rev1 DF architectures. The efficiency obtained from the Rev2, Rev1, and Rev0 DF architectures for same AER packet generation rate are 96%, 58%, and 12% respectively.

The router sub-module in the APP_Router in a 1-dimension (Rev1) and 2D torus network (Rev2) calculates the shortest routing path and transmits the AER

packet to the destination. For the DF architectures defined here, AER packets are dropped during packet/link contention, greatly simplifying the protocols and reducing the complexity of designing and implementing the system. The simulation results indicate that the Rev2 DF architecture system would perform better and provide a baseline architecture for further revisions. Future revisions of the architecture will be built upon on the Rev2 DF architecture and will address the issues observed with the Rev2 DF architecture.

Chapter 8

Conclusion & Future Work

8.1 Conclusion

In conclusion, we have built message passing network model simulators, which integrate spiking analog cortical modules (APP's). The inter-APP communication protocol employed is the Address Event Representation (AER)[4]. We implemented the DF architecture in revisions namely Rev0, Rev1 and Rev2, adding more complexity in each iteration. We have explored the different architecture schemes single bus arbiter, 1-dimensional token ring and 2D torus network at each revision stage. To pass the control and data between the router sub-modules in 1-dimension token ring network and 2D torus network we have used channels formed with the help of Ambric registers. Each Ambric register is preemptive, in that its operation is self-initiated. The protocol is simple, using only two signals and no intermediate logic is required.

For the work discussed here, the basic architectures are implemented. The Rev0, Rev1, and Rev2 DF architectures are implemented using Verilog. Simulation of the Verilog designs is performed using Modelsim. The FPGA board used for demonstration purposes is the Xilinx Spartan 3E Starter Kit[14]. The statistics obtained in each revision are used to analyze the network and give confidence that the 2D

torus, AER packet based network is an appropriate communication architecture for the SyNAPSE system. Future systems will build on the basic results presented here.

With the help of results obtained from the simulations, the Rev2 DF architecture proves to have better performance with respect to total packets lost in the network. The total packets lost in the network for the Rev2 DF architecture is much lower in comparison to Rev0 and Rev1 DF architectures. The results obtained from the simulations lays a foundation for further revisions.

8.2 Future Work

The Digital Fabric (DF) is developed in stages. The work presented here includes 3 revisions to the network architecture. For future revisions, the complexity of packet generation will be increased. We can create a high level C++ simulation that can more rapidly model network trade-offs, etc. for performance tuning and architecture trade-off studies. This will be very useful as we start to dial in the actual connectivity requirements of the algorithms. Also, we need to understand the connectivity requirements of the various neural architectures being considered. Ideally some kind of automatic connection generation tool would be very helpful. Another very important issue concerns the mapping of the models to the combined APP/DF system. We need to determine the model connectivity requirements, and whether the architecture can handle those satisfactorily. Plus it is likely that some architectural features will need to be added to the DF for handling certain unique characteristics of some of the models. For example, models that have spike bursts or those that have significantly local axon bifurcation may benefit from

specialized packet functionality. Also, the number and size of the address maps that are required may be a problem. Every connection that goes outside of an APP requires a physical address. There are a variety of ways some of those addresses can be optimized, such as regional broadcast, which trade-off DF hardware complexity for address bits.

References

- [1] C. Mead, “*Analog VLSI and neural systems*,” VLSI systems series, Addison-Wesley, 1989.
- [2] K. A. Boahen, “A throughput-on-demand address-event transmitter for neuromorphic chips,” *Conference on Advanced Research in VLSI*, 0:72, 1999.
- [3] D. Hammerstrom, “*A connectionist/neural network bibliography*,” Technical report, Oregon Graduate Center, 1986.
- [4] K. A. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(5):416–434, May 2000.
- [5] M. Mahowald, “*VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*,” PhD Dissertation, California Institute of Technology, Pasadena, CA, USA, 1992.
- [6] M. A. Sivilotti, “*Wiring considerations in analog VLSI systems, with application to field-programmable networks*,” PhD Dissertation, California Institute of Technology, Pasadena, CA, USA, 1991. UMI Order No. GAX91-37292.

- [7] M. Mahowald, “*An analog VLSI system for stereoscopic vision*,” Kluwer international series in engineering and computer science, Kluwer Academic Publishers, 1994.
- [8] R. C. Dixon, N. C. Strole, and J. D. Markov, “A token-ring network for local data communications,” *IBM Systems Journal*, 22(1.2):47–62, 1983.
- [9] J. D. William and L. S. Charles, “The torus routing chip,” *Journal of Parallel and Distributed Computing*, 1(4):187–196, 1986.
- [10] W. C. Barry and J. G. Michael, “Intrinsic firing patterns of diverse neocortical neurons,” *Trends in Neurosciences*, 13(3):99–104, 1990.
- [11] G. B. Ermentrout and N. Kopell, “Parabolic bursting in an excitable system coupled with a slow oscillation,” *SIAM J. Appl. Math.*, 46:233–253, April 1986.
- [12] A. G. Melvyn and A. D. Milner, “Separate visual pathways for perception and action,” *Trends in Neurosciences*, 15(1):20–25, 1992.
- [13] S. S. Haykin, “*Neural networks: a comprehensive foundation*,” Prentice Hall, 1999.
- [14] Xilinx Corporation, “*Xilinx Spartan 3E Family Student Starter Kit*,”
- [15] P. P. Chu, “*FPGA prototyping by Verilog examples: Xilinx Spartan -3 version*,” J. Wiley & Sons, 2008.
- [16] T. Kohonen, “*Content-addressable memories*,” Springer series in information sciences, Springer-Verlag, 1987.

- [17] L. Chisvin and R. J. Duckworth, “Content-addressable and associative memory: alternatives to the ubiquitous RAM,” *Computing Processing (Hardware/Software)*, 22(7):51–64, Jul 1989.
- [18] K. E. Grosspietsch, “Associative processors and memories: a survey,” *IEEE Micro*, 12(3):12–19, Jun 1992.
- [19] I. N. Robinson, “Pattern-addressable memory,” *IEEE Micro*, 12(3):20–30, Jun 1992.
- [20] S. Stas, “Associative processing with cams,” *Conference Record Northcon/93*, pages 161–167, Oct 1993.
- [21] M. Butts, A. M. Jones, and P. Wasson, “A structural object programming model, architecture, chip and tools for reconfigurable computing,” *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007, FCCM 2007*, pages 55–64, April 2007.
- [22] Xilinx Corporation, “*Microblaze 32-bit RISC Soft Core Processor*,” 2006.
- [23] Mentor Graphics, “*Modelsim Mentor Graphics Student PE Edition 6.5d*,” 1981.

Appendix A

Rev2 APP_Router Sub-Module Input/Ouput Signals Description

A.1 Signals Description used in Figure 5.2 and 5.4

A.1.1 Figure 5.2 Signals

- R0 - Right Channel receiving data from node [0,0]
- L0 - Left Channel receiving data from node [0,0]
- D0 - Down Channel receiving data from node [0,0]
- U0 - Up Channel receiving data from node [0,0]

R, L, D and U denotes left, right, down and up channel, number denotes from which node the channel is receiving the data.

A.1.2 Figure 5.4 Signals

- AR-R0 - Right Channel receiving data from node [0,0].
- AR-L0 - Left Channel receiving data from node [0,0].
- AR-D0 - Down Channel receiving data from node [0,0].
- AR-U0 - Up Channel receiving data from node [0,0].

- AR-L1 - Data coming to node $[0,0]$ from node $[0,1]$ through the left channel.
- AR-R2 - Data coming to node $[0,0]$ from node $[0,2]$ through the Right channel.
- AR-U3 - Data coming to node $[0,0]$ from node $[1,0]$ through the Up channel.
- AR-D6 - Data coming to node $[0,0]$ from node $[2,0]$ through the Down channel.
- A - FIFO_Write
- B - FIFO_Empty
- C - FIFO_FULL
- D - FIFO_Read
- E - Valid_Out_Channel
- F - Valid_Out_Channel
- G - Accept_In_Channel
- H - Accept_In_Channel
- I - Valid_Router_In_Right_Channel
- J - Valid_Router_In_Left_Channel
- K - Accept_Router_Out_Right_Channel
- L - Accept_Router_Out_Left_Channel
- M - Accept_In_Channel

- N - Valid_Out_Channel
- O - Valid_Router_In_Up_Channel
- P - Accept_Router_Out_Up_Channel
- Q - Accept_In_Channel
- R - Valid_Out_Channel
- S - Accept_Router_Out_Down_Channel
- T - Valid_Router_In_Down_Channel
- U - Destination_Match
- V - App_Out_From_CAM

AR stands for Ambric Register. R, L, D and U denotes left, right, down and up channel, number denotes from which APP node it is receiving the data.

Appendix B

Rev0, Rev1 & Rev2 Modelsim Simulation Waveforms

B.1 Rev0 Results: Arbitration model

Figure B.1 is a screen shot of the simulations performed for the Rev0 DF architecture, operating in Broadcast mode. In Broadcast mode, the AER packet generated has to be transmitted to all the APP_In sub-module. The 32 bit register Bus_req_from_all_App_Out in the screen shot, represents the bus request from APP_Out31 - APP_Out0 sub-module i.e 1st bit in 32 bit register Bus_req_from_all_App_Out represents request from the App_Out0 sub-module at the given clock cycle. At 3770ns (nano second) we can see that more than one APP_Out sub-module is sending the request to the Bus_Control sub-module to process the AER packet. In the simulation, the priority is set to lower sub-modules i.e APP_Out0 has higher priority over APP_Out1, APP_Out1 has higher priority over APP_Out2 and so on. Thus at 3770ns APP_Out1 has the higher priority over other APP_Out sub-modules and bus grant is provided to the APP_Out1 sub-module in the next clock cycle (1st bit of Bus_granted_to_only_one_APP_Out becomes active high at 3790ns). After receiving the bus grant from the Bus_Control sub-module, the APP_Out1 sends the data out in the next clock cycle (at 3810ns output we can see change in Data_Out_From_APP_Out1). The Bus_Control sub-module reads

the packet in 2 clock cycles (3830ns and 3850ns in the screen shot), and since the operating mode is Broadcast, this packet is sent to all the APP_In sub-modules. Data_In_App_In[0] - Data_In_App_In[31] represents the input at the APP_In sub-module.

B.1.1 Broadcast Mode

Figure B.1: Rev0 DF Architecture Simulation Results Broadcast Mode

B.1.2 Goto Mode

Figure B.2 shows the screen shot of the simulations, performed for the Rev0 DF architecture operating in Goto mode. In Goto mode, the AER packet generated has to be transmitted to one single APP_In sub-module. In the screen shot App_Out0_Data_Out - App_Out31_Data_Out represents the data being transmitted from APP_Out sub-module upon the bus grant from the Bus_Control sub-module. At 4090ns we can see that all 32 bits of Bus_Req_From_All_APP_Out module are asserted. This indicates that all the APP_Out sub-modules (0 -31) is sending the request to the Bus_Control sub-module to transmit it's AER packet. In the next clock cycle 4110ns bus grant is given to the APP_Out0 sub-module, since it has the higher priority over the other APP_Out sub-module. After receiving the bus grant from the Bus_Control sub-module, the APP_Out0 sub-module then sends out the AER packet (at 4130ns in the screen shot). The Bus_Control sub-module reads this AER packet and transmits it in 2 clock cycles (4150ns and 4170ns). The APP_In sub-module keeps snooping on the Bus_Control sub-module for transactions. Since the operation mode is Goto, the AER packet is destined to only one of the APP_In sub-module. The last 5 bits in the AER packet determines the destination. In the screen shot we can see that, last 5 bits of the AER packet contains value 7 (binary value). At 4170ns, we can see that a match is found in APP_In7 sub-module via it's internal CAM. After the AER packet reaches the destination, one of the bit out of 31 bits in Pulse_Out is asserted depending on the value of the wire address in the AER packet (i.e. 9:5 bits in the AER packet).

Figure B.2: Rev0 DF Architecture Simulation Results Goto Mode

Waveform B.3 and B.4 show screen shot for the Rev0 DF architecture simulation for different pulse generation rates. In the screen shot B.3 we can see that pulse generation rate is lower, hence lesser AER packets are generated and the interval between the AER packet generation is also more. Since, the interval and the packet generation rate is relaxed, the FIFO is empty for longer clock cycle duration, thus reducing the total number of AER packets lost in the network. The registers APP_Out0_Packets_Lost - APP_Out31_Packets_Lost represent the counters inside each APP_Out sub-module and keeps a count on the packet lost due to the FIFO being full. The registers “Total AER Packets Generated” and “Total AER Packets Lost” is sum of all the individual AER packets generated and the AER packets lost respectively, in the network. In the screen shot B.4 the pulse generation rate is higher, hence more AER packets are generated at very less interval leading to more AER packet loss in the network.

Figure B.3: Rev0 Simulation Results:Arbitration Model:Lower Packet Generation Rate

Figure B.4: Rev0 Simulation Results:Arbitration Model:Higher Packet Generation Rate

Modelsim Student Edition PE 6.5d is used for verilog simulation. Below sections show screen shots for Broadcast and Goto mode obtained with the help of Modelsim.

B.1.3 Rev1 Broadcast Mode

Figure B.5 is a screen shot of the simulation performed in Broadcast mode. In Broadcast mode, the AER packets generated has to be transmitted to all the nodes in the network. The signal name App_Out0_Data_Out in the screen shot represents the AER packet, read by each node from it's FIFO sub-module. The FIFO, stores only the AER packets generated by the AER/Convert sub-module. The AER packets coming from neighboring nodes are not stored but transmitted immediately or dropped if the FIFO is full. The signal name App_In0_Data_In - App_In31_Data_In in the screen shot represents the data input to the APP_In sub-module. In the Broadcast mode, the AER packet format is as below:

- Bits 15-10 : Operation mode , for Broadcast value is 000010
- Bits 9-5 : Wire address
- Bits 4-0 : Source address

In the above screen shot, at 2130ns, we can see the AER packet information. Since, the operation mode is set to Broadcast, we can see that at every clock cycle each node decodes the AER packet and sends it to it's respective APP_In sub-module for wire address matching, and also transmits the AER packet to its neighboring node. The signal name App_In0_Cam_Found_Match - App_In9_Cam_Found_Match is asserted, when there is a match, and subsequently

a pulse is generated

(App_In0_Pulse_Out - App_In9_Pulse_Out).

Figure B.5: Rev1 DF Architecture Simulation Results Broadcast Mode

B.1.4 Rev1 Goto Mode

Figure B.6 represents the screen shot of the simulation performed in Goto mode. In Goto mode, the AER packet generated has to be transmitted to only one node. Bit 4:0 in the AER packet contain the destination node address. The signal name App_Out0_Data_Out in the screen shot represents the AER packet, read from the FIFO sub-module. The FIFO stores only the AER packets generated by the AER/Convert sub-module. In the screen-shot, between 3600ns - 3800ns, we can see the router sub-module reads the data from the FIFO and then sends the AER packet either to the left or to the right channel. The signal name App_In0_Data_In - App_In31_Data_In represents the AER packet input to the APP_In sub-module. At 3800ns, we can see App_In12_Data_In receives the AER packet, since AER packet[4:0] value is 12(5'b01100). The signal App_In12_Cam_Found_Match is asserted, as there is a match in the CAM and subsequently a pulse is generated (App_In12_Pulse_Out). At 4800ns, we can see AER packets being dropped by each node due to link contention.

Figure B.6: Rev1 DF Architecture Simulation Results Goto Mode

Modelsim Student Edition PE 6.5d is used for Verilog simulation. Below sections show the screen shots for the Broadcast and the Goto mode obtained with the help of Modelsim.

B.1.5 Rev2 Broadcast Mode

Figure B.7 represents the screen shot of the simulation performed in Broadcast mode. In Broadcast mode, the AER packet generated has to be transmitted to all the nodes in the network. The signal name `Data_Input_Router0`, in the screen shot represents the AER packet, read by the router sub-module from the FIFO sub-module. The FIFO stores only the AER packets generated by the AER/Convert sub-module. The AER packets coming from the neighboring nodes are not stored but transmitted immediately or dropped if the FIFO is full. The signal name `App_In0_Data_Input - App_In31_Data_Input`, in the screen shot represent the data input to the APP_In sub-module. In the Broadcast mode, the AER packet format is as below:

- Bits 15-10 : Operation mode , for Broadcast value is 000010
- Bits 9-5 : Wire address
- Bits 4-0 : Source address

In the screen shot above at 1600 ns we can see the AER packet information. The sub-module APP0 is responsible for generating the AER packet shown in the screen shot, hence bits 4 0 in the AER packet is filled with 0 (source address). Since, the operation mode is Broadcast, we can see that at every clock cycle each router sub-module decodes the AER packet and sends it to its respective APP_In for wire

address matching, and also transmits the AER packet to its neighboring nodes. If there is match a pulse is generated ($\text{App_In0_Pulse_Out} - \text{App_In21_Pulse_Out}$).

Figure B.7: Rev2 DF Architecture Simulation Results Broadcast Mode

B.1.6 Rev2 Goto Mode

Figure B.8 shows the screen shot of the simulation, performed in Goto mode. The AER packet generated has to be transmitted to only one node. In the screen shot, Pulse_Generator_Output represents the output of the APP sub-module and is input to the Packet_Generator sub-module (AER/Convert). The Pulse_Generator_Clock, the Packet_Generator_Clock, the Router_Clock represents the operating frequency of the Pulse_Generator sub-module (APP), the Packet_Generator sub-module (AER/Convert), and the router sub-module respectively. At 3000ns, Packet_Generator sub-module (AER/Convert) receives 32 bit input (signal name is Packet_Generator_Input_From_Pulse_Generator_Module) from the Pulse_Generator sub-module (APP). After receiving the input, the Packet_Generator sub-module (AER/Convert) generates the AER packets at every clock cycle (here 4 AER packets are generated as 4 bits out of 32 bits are high at the input) and stores the AER packet in the FIFO. Depending upon the availability of the channel, the router sub-module reads the AER packet from the FIFO. At 3300ns, we can see the router sub-module reads the AER packets from the FIFO and steers the AER packet to the neighboring node.

We can also see how the router sub-module makes decision and routes the AER packet in the network. Since the operation mode is Goto, Bits 4:0 in AER packet represents the destination node address. For simulation purpose, the node is arranged as 4x8 torus network. At 2400ns, the router sub-module in the node [0,0], reads the AER packet from its FIFO sub-module and has to route it to destination node [4,1] (value of the destination node is 01110 -14). The network route taken in this 4x8 network is node [0,0] to [1,0], [1,0] to [2,0], [2,0] to [3,0] via the down

channel, and then [3,0] to [3,3], [3,3] to [3,2] via the left channel (between 2950ns - 3100ns in the screen shot). After reaching the destination i.e node [3.2], the router sub-module routes the AER packet to it's APP_In sub-module. The APP_In sub-module checks its internal data with the incoming AER packet via CAM, and upon match in the CAM, a pulse is generated (one of the bit in APP_In_Pulse_Out is asserted at 3150ns).

The signal name App_Out0_Data_Out in the screen shot represents the AER packet, read from the FIFO sub-module. The FIFO stores only the AER packets generated by the AER/Convert sub-module. Between 3600ns - 3800ns the node reads the data from the FIFO and then sends the AER packet either to the left or the right channel or the up channel or the down channel. The signal name App_In0_Data_In - App_In31_Data_In represents the AER packet input the APP_in sub-module. At 3800ns, App_In12_Data_In receives the AER packet, since AER packet [4:0] value is 12 (5'b01100). The signal name App_In12_Cam_Found_Match is asserted, as there is match in the CAM and subsequently a pulse is generated (App_In12_Pulse_Out). At 4800ns AER packets being dropped by each APP-DF node due link contention.

Figure B.8: Rev2 DF Architecture Simulation Results Goto Mode