An FPGA Implementation of a High Performance AER Packet Network

Sirish Kumar Munipalli
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

Recommended Citation

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.
An FPGA Implementation of a High Performance AER Packet Network

by

Sirish Kumar Munipalli

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
Roy Kravitz
Douglas V. Hall

Portland State University
2013
Abstract

This thesis presents a design to route the spikes in a cognitive computing project called Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE). SyNAPSE is a DARPA-funded program to develop electronic neuromorphic machine technology that scales to biological levels.

The basic computational block in the SyNAPSE system is the asynchronous spike processor (ASP) chip. This analog core contains the neurons and synapses in a neural fabric and performs the neural and synaptic computations. An ASP takes asynchronous pulses (spikes) as inputs and after some small delay produces asynchronous pulses as outputs. The ASP chips are organized in a nxn (where $n \approx 10$) 2-dimensional grid with a dedicated node for each chip. This interconnected network is called Digital Fabric (DF) and the node is called Digital Fabric Node (DFN). The DF is a packet network that routes pulse (AER - Address event representation) packets between ASP’s.

This thesis also presents a technique for design implementation on a FPGA, performance testing of the network and validation of the network using various tools.
Acknowledgements

I would like to thank my academic and thesis advisor Dr. Dan Hammerstrom for guiding me through this research work and also for providing me with an opportunity to be a part of Biologically inspired computing lab as an Graduate Research Assistant. I would also like to thank my friend Danny Voils for helping me on this research and his C++ simulator was helpful in making architectural decisions for this design.

I would also like to thank HRL labs for funding me through this research.

I am also grateful to the committee members, Dr. Douglas V. Hall and Prof. Roy Kravitz for reviewing this document and suggesting key changes.

Finally, I would like to thank my family and friends for supporting me.
Contents

Abstract

Acknowledgements

List of Tables

List of Figures

1 Introduction

1.1 About SyNAPSE .............................................. 1
1.2 Contribution of this work .................................. 2
1.3 Block diagram of Digital Fabric subsystem ............... 2
1.4 Thesis organization ........................................ 3

2 DF system overview

2.1 System level architecture of Digital Fabric Node ....... 4
2.2 Circular FIFO .............................................. 5
2.3 Network Interface Unit .................................... 5
2.4 Router Unit ................................................. 5
2.5 Routing Table ............................................... 5
2.6 Spike Packet Unit ......................................... 6
2.7 Asynchronous spike packet Interface Unit ............... 6
6 Packet Trace Capabilities

6.1 Debug, Performance

6.1.1 Debugging in simulated environment

6.1.2 Verilog Simulation

6.1.3 Packing the data into a Matlab array data structure using Perl

6.1.4 Data analysis using Matlab

6.1.5 Matlab packet tracking algorithm

6.2 Real-time Debug

6.3 Derivation for predicting the Path Cost

6.3.1 Path cost for simulation

6.3.2 3D plot of $P_{c \text{sim}}$

6.4 Results

6.4.1 Wave diagrams

6.4.2 Simulation results

6.4.3 Jitter Calculation

6.4.4 Synthesis Report

6.5 Self tracing

6.5.1 Jitter calculation technique

6.5.2 Advantages of using self tracing and back tracing approach

7 Conclusion and Future work

7.1 Conclusion
7.2 Future work ................................................. 63

References .............................. 65

Appendices ............................... 68

A Matlab Variables .......................... 69
   A.1 APP(1,clock_instant,x,y) ................. 69
   A.2 DFN_X_SEND(1,clock_instant,x,y) ......... 69
   A.3 DFN_Y_SEND(1,clock_instant,x,y) ......... 69
   A.4 DFN_dir_e(1,clock_instant,x,y) .......... 70
   A.5 DFN_dir_s(1,clock_instant,x,y) .......... 70
   A.6 DFN_packet_track(1,clock_instant,x,y) .. 70
   A.7 DFN_fifo(1,clock_instant,x,y,dir_niu,in_out,head_tail) .. 70
   A.8 DFN_NIU(1,clock_instant,x,y,dir_niu,in_out,head_tail) .. 71

B Petalinux .................................. 73
   B.1 Petalinux Environment Setup .............. 73
   B.2 Rebuilding the reference design .......... 73
   B.3 Testing the software image with QEMU .... 76
   B.4 Testing the image on hardware .......... 76
   B.5 Using C-Kermit ......................... 77
   B.6 C code for reading a file ............... 78

C Xilinx EDK Design suite ................. 79
   C.1 Setting Xilinx environment .............. 79
   C.2 Building a Hardware project for MicroBlaze .... 79
D  Downloading the Bit Image to the FPGA and Petalinux server  85
   D.1  Installing the USB drivers  ...................................................  85
   D.2  Downloading the bit file to the FPGA using iMPACT .................  86
   D.3  Petalinux webserver ...............................................................  86

E  Wave Diagram  88
   E.1  Wave Diagram .................................................................  88

F  Perl Regex  90
   F.1  Perl Regular expressions (Regex) ........................................  90
       F.1.1  Matching a string .....................................................  90
       F.1.2  Wildcards and Repetitions .........................................  91
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Estimate of data gathered for a 10x10 design</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>Maximum and minimum path costs for different values of RR</td>
<td>54</td>
</tr>
<tr>
<td>6.3</td>
<td>Simulation results for different packet rates for a 4x4 network</td>
<td>55</td>
</tr>
<tr>
<td>6.4</td>
<td>Simulation results for different packet rates for a 10x10 network</td>
<td>56</td>
</tr>
<tr>
<td>6.5</td>
<td>Jitter calculations for 3 different packet generation rates for a 4x4 network</td>
<td>58</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Block diagram of the system .................................................. 2

2.1 A block diagram of a Digital Fabric Node (DFN) .................... 4

3.1 The DF routes spikes between ASP’s ................................. 7

3.2 A simplified schematic of Interconnected nodes .................... 8

3.3 Schematic of the digital fabric network indicating the number of
     possible paths to reach the destination. The red node indicates the
     source and the green node indicates the destination. ............. 9

4.1 A schematic of microarchitecture of the Node(DFN).  ABS
     registers are temporary registers for arithmetic
     calculations. The ext bus indicates the external bus connecting the respective neighboring node and has a
     bus width equal to Packet Size as indicated in the above figure. Refer figure 4.3 for I/O connections ... 13

4.2 Schematic of Circular FIFO .................................................. 15

4.3 A interconnection layout between the adjacent NIU’s .............. 17

4.4 Schematic of communication between two Network Interface Units
     (NIU) ............................................................................... 18

4.5 Schematic of communication between two Network Interface Units
     (NIU) ............................................................................... 19

4.6 Layout of the routing table .................................................... 20

4.7 Schematic of the AIU used in this design .............................. 21

4.8 Hardware implementation of the snapshot approach ............. 22

4.9 Hardware implementation of the multiplexer approach ......... 23

4.10 Hardware implementation of the serial approach ............... 24
B.3 Petalinux compilation progress ........................................ 76

C.1 XPS screenshot showing the bitstream generation .................. 80
C.2 XPS screenshot for exporting the design to SDK .................... 81
C.3 Softcore processor template generation ............................... 83
C.4 Screenshot of the bitstream generation and updating the bit file with
the processor data in ISE ................................................. 84

D.1 A screenshot from xilinx ISE for launching the iMPACT tool ...... 86
D.2 A screenshot of petalinux webpage .................................... 87

E.1 Wave diagram of a transaction between adjacent nodes. The labels
surrounded in the red box indicates a track for simulation purpose
only ................................................................. 89
Chapter 1

Introduction

1.1 About SyNAPSE

Scalability and connectivity are two key challenges in designing neuromorphic hardware that can match the mammalian brain. SyNAPSE is a DARPA program that aims to develop electronic neuromorphic machine technology that scales to biological levels. The acronym SyNAPSE stands for Systems of Neuromorphic Adaptive Plastic Scalable Electronics.

The basic computational block in the SyNAPSE system is the asynchronous pulse processor (ASP) chip. This analog core contains the neurons and synapses in a neural fabric and performs the neural and synaptic computations. ASP chips are organized in an $n \times n$ 2-dimensional grid with a dedicated node for each chip. This interconnected network is called Digital Fabric (DF) and the node is called Digital Fabric Node (DFN).

The challenge in scalability is to implement $10^6$ neurons and $10^{10}$ synapses with an average fan-out of $10^4$, in a square centimeter of CMOS. The challenge in connectivity is to connect these $10^{10}$ synapses [1]. Recently nanotechnology [14][6] has been integrated with CMOS to achieve the required synaptic density.

In this thesis I present a design scheme for routing spikes generated by the ASP. The ASP takes the asynchronous pulses as inputs and after a small delay it produces asynchronous pulses as output. Depending upon the arrival time of the pulses the weights are set in the ASP chips. These weights determine the firing of
the neuron depending upon the threshold.

1.2 Contribution of this work

My role in this project was to design the DF for routing the spikes generated by the ASP to other ASP’s. In this project I have designed an ASP architecture that allows DF scaling, including the problem for routing the packets in all four directions and designed a method to effectively validate the entire design from the top level.

1.3 Block diagram of Digital Fabric subsystem

![Figure 1.1: Block diagram of the system](image)

Depicted above is the system block diagram which was implemented on a
Spartan-6 FPGA [18]. The routing tables are loaded from the host PC into the Block RAM (BRAM) [19] and when the loading is done, the system is initiated and the DFN starts reading from this BRAM as the spikes arrive from the ASP. For larger systems it may be necessary to interface to a external DDR RAM and use the dual ported BRAM as a cache. This approach is not considered here.

1.4 Thesis organization

Chapter 2 provides the basic system overview and a brief description on the internal modules of the design. Chapter 3 talks in detail about the routing algorithm and the DF layout. It has an example of the routing algorithm. Chapter 4 provides an insight into the microarchitecture of the system in detail and some performance bottlenecks. Chapter 5 discusses how the various tools and programming languages were used to implement the design. Apart from this it also describes a method for interacting with the all the nodes in the DF. Chapter 6 describes a method for debugging the system at the top level and generating the network statistics from the simulation log files. Chapter 7 concludes the thesis by stating the identified performance bottlenecks and suggesting new features that can improve the design.
Chapter 2

DF system overview

2.1 System level architecture of Digital Fabric Node

![Block Diagram of Digital Fabric Node](image)

Figure 2.1: A block diagram of a Digital Fabric Node (DFN)

Depicted above is the top level architecture of the system. The yellow wires represent the debug lines and the brown wire from the soft-core processor loads the tables into the memory. In addition to loading the tables the soft-core processor is also used to communicate with the host PC and to retrieve the debug log.
2.2 Circular FIFO

Since the system uses STDP [12] [4] spike trains we can ignore the packets which do not reach the destinations in time. For these kinds of situations a circular FIFO is an ideal implementation as we do not need to worry about clearing the FIFO once the FIFO is full. Hence the oldest packets are dropped first, which is acceptable in our system.

2.3 Network Interface Unit

The network Interface unit (NIU) is used to communicate between the adjacent nodes. Each NIU consists of two FIFO blocks, one for the incoming packets and one for the outgoing packets. An AMBRIC register [10] communication model is used for communicating with the adjacent NIU’s.

2.4 Router Unit

The router unit is in the node is responsible for routing the packets in all the four directions in a systematic manner. The router uses a round robin algorithm for scheduling the packets coming from Spike Packet Unit (SPU) and the four NIU’s. This algorithm gives equal priority to all the units and equal intervals of time slots. An ACCEPT TO RECEIVE signal is asserted by the router when a NIU or SPU is selected by the scheduler.

2.5 Routing Table

The routing table is a large look-up table that stores the mapping between the source ASP output pins and the destination router node input pins. In a complete
system the mapping will be between the source and the destination pins, instead of the router node address.

2.6 Spike Packet Unit

The Spike Packet Unit (SPU) is responsible for creating a packet. It uses information from the routing table module which maps the ASP pin number to the destination address.

As this is a simulation model, the routing tables can be of any size, but in reality they are limited by implementation requirements. The required size depends on the neural models being emulated.

2.7 Asynchronous spike packet Interface Unit

The asynchronous spike packet interface unit (AIU) is used to generate the pin number when a spike is generated at its input. Basically the AIU is a large encoder whose pins can be limited by the number of I/O pins present on the FPGA chip. Since this is a simulation model any number of ASP pins can be interfaced to the AIU module. The ASP pin number is used as an index to look up the destination node address in the routing table.
Chapter 3

Routing

3.1 Introduction to routing

The routing table provides the flexibility to alter the specific synaptic connections between the neurons. For simulating the actual system, routing tables will be generated by a neuromorphic compiler [7]. For this simulation the routing tables were randomly generated by using LFSR registers. The routing module used in this design is simple and uses simple round robin scheduling with equal priority for routing the packets in the specified directions.

All the nodes are arranged in a grid format with equal $x$ and $y$ dimensions. The ends of the grid have been wrapped around to create a torus connectivity to reduce average routing distance.

![Figure 3.1: The DF routes spikes between ASP’s](image)

Figure 3.1: The DF routes spikes between ASP’s
3.2 Node interconnections

Figure 3.2: A simplified schematic of Interconnected nodes

The interconnections between the nodes play an important role in routing the packets at a higher speed. Depicted in the above figure is a $16 \times 16$ digital fabric network, each node $N_{(x,y)}$ represents a DFN. Note that the edges in the network are wrapped around both along the $x$, $y$ directions to create a torus. For example on the x-axis, the west NIU of $N_{(1,4)}$ is connected to the east NIU of $N_{(8,4)}$, and on the y-axis the north NIU of $N_{(1,1)}$ is connected to south NIU of $N_{(1,8)}$.  

8
3.3 Routing algorithm

Figure 3.3: Schematic of the digital fabric network indicating the number of possible paths to reach the destination. The red node indicates the source and the green node indicates the destination.

The routing algorithm described here has been written with the hardware implementation of the router in mind. The first step in this algorithm involves the calculation of the absolute distance along the x and the y directions. As there is no direct data type for Verilog standards older than Verilog-2001, to implement
signed subtraction it needs to be agreed upon a fixed data size of the register variables on which the arithmetic operations are going to be performed. The negative output can be determined by taking an extra MSB to indicate the sign. While calculating the difference between the current \( x \) coordinate and the destination \( x \) coordinate if the MSB of the result is 1, the packet must be routed in the west direction else in the east direction. Similarly, for the \( y \) direction in the north and south directions respectively.

As mentioned before, the network connectivity plays an important role in the routing speed of the packets. The torus connectivity [16] enables us to calculate the reverse distance from the current location to the destination location. The distance along the torus path is denoted by \( \Delta x_{\text{reverse}} \) for the \( x \) direction and \( \Delta y_{\text{reverse}} \) for \( y \) direction and the Manhattan distances with respect to the axis shown in the above figure along \( x \), \( y \) directions are indicated by \( \Delta x \), \( \Delta y \) respectively.

### 3.3.1 How does the router choose between \( x \), \( y \) directions?

If \( \Delta x \) is less than \( \Delta y \) then the packet will be routed along the \( x \) direction first else the packet will be routed along the \( y \) direction. If \( \Delta x \) was less than \( \Delta y \), a comparison between \( \Delta x \) and \( \Delta x_{\text{reverse}} \) will be done and the packet will be routed along the direction with the least magnitude and a similar selection is made for \( y \) direction if \( \Delta y \) was less than \( \Delta x \).

If \( \Delta x \) equals zero then the packet has reached the destination \( x \)-coordinate and it should travel only along the \( y \) direction. Similarly the packet needs to travel only along the \( x \) direction until it reaches the destination if \( \Delta y \) equals zero. When \( \Delta x \), \( \Delta y \) are not equal to zero and \( \Delta x \) equals \( \Delta y \) then the direction to route the packet is chosen randomly to balance the load.
3.3.2 Example

The algorithm here is explained with respect to the *fig(3.2)*.

\[
\Delta x = D\{N_{(3,8)},N_{(4,8)},N_{(5,8)},N_{(6,8)},N_{(7,8)},N_{(8,8)}\}
\]

\[
\Delta y = D\{N_{(3,4)},N_{(3,5)},N_{(3,6)},N_{(3,7)},N_{(3,8)}\}
\]

\[
\Delta x_{\text{reverse}} = D\{N_{(3,8)},N_{(2,8)},N_{(1,8)},N_{(8,8)}\}
\]

\[
\Delta y_{\text{reverse}} = D\{N_{(3,4)},N_{(3,3)},N_{(3,2)},N_{(3,1)},N_{(3,8)}\}
\]

From the above example it can be seen that \(\Delta x\) is greater than \(\Delta y\) therefore the packet is routed along the y-axis first until \(\Delta y\) is equal to zero and then along x-axis. Since \(\Delta x_{\text{reverse}}\) has fewer hops when compared to \(\Delta x\), the packet will be routed west instead of east, saving one hop.
Algorithm 1 Algorithm for routing the packets

loop
    if posedge clk then
        $\Delta x_{\text{sign}} \leftarrow ((x_{\text{dest}} - x_{\text{current}}) > 0) ? 0 : 1$
        $\Delta y_{\text{sign}} \leftarrow ((y_{\text{dest}} - y_{\text{current}}) > 0) ? 0 : 1$
        if $\Delta x_{\text{sign}} = 0$ then
            $\Delta x \leftarrow (x_{\text{dest}} - x_{\text{current}})$
            $\Delta x_{\text{reverse}} \leftarrow (\text{Width of the GRID} - x_{\text{dest}}) + x_{\text{current}}$
            if $\Delta x \leq \Delta x_{\text{reverse}}$ then
                Route the packet EAST
            else
                Route the packet WEST
            end if
        else
            $\{\Delta x_{\text{sign}} \neq 0\}$
            $\Delta x \leftarrow (x_{\text{current}} - x_{\text{dest}})$
            $\Delta x_{\text{reverse}} \leftarrow (\text{Width of the GRID} - x_{\text{current}}) + x_{\text{dest}}$
            if $\Delta x \leq \Delta x_{\text{reverse}}$ then
                Route the packet WEST
            else
                Route the packet EAST
            end if
        end if
    end if
    if $\Delta y_{\text{sign}} = 0$ then
        $\Delta y \leftarrow (y_{\text{dest}} - y_{\text{current}})$
        $\Delta y_{\text{reverse}} \leftarrow (\text{Height of the GRID} - y_{\text{dest}}) + y_{\text{current}}$
        if $\Delta y \leq \Delta y_{\text{reverse}}$ then
            Route the packet SOUTH
        else
            Route the packet NORTH
        end if
    else
        $\{\Delta y_{\text{sign}} \neq 0\}$
        $\Delta y \leftarrow (y_{\text{current}} - y_{\text{dest}})$
        $\Delta y_{\text{reverse}} \leftarrow (\text{Height of the GRID} - y_{\text{current}}) + y_{\text{dest}}$
        if $\Delta y \leq \Delta y_{\text{reverse}}$ then
            Route the packet NORTH
        else
            Route the packet SOUTH
        end if
    end if
end loop
4.1 Microarchitecture of the Node

Figure 4.1: A schematic of microarchitecture of the Node (DFN). ABS registers are temporary registers for arithmetic calculations. The ext bus indicates the external bus connecting the respective neighboring node and has a bus width equal to Packet Size as indicated in the above figure. Refer figure 4.3 for I/O connections.
The microarchitecture is designed to be implemented by a small size FPGA and has been optimized for maximum speed. In reality the AIU unit can be more complex as there can be an extremely large number of pins on the ASP which need to connect to this unit, this requires a multiplexing unit between the ASP and the AIU in order to accommodate all the pins. The ASP receives and generates the asynchronous pulses based on the weights of the neurons. The APP unit interfaces to the DFN unit through the AIU unit. The SPU and the router blocks have been separated so that the routing algorithm can be replaced by a more intelligent algorithm in the future. Once the pin number from the AIU is generated, it is fed to the SPU. The SPU unit will match the pin number to the destination address. The routing table in the SPU contains the mapping to the pin number and the destination address. The generated packet is supplied to the router when the round robin scheduler allocates a time slot to the respective packet. The round robin scheduler allocates a total of 5 different time slots - 4 for the 4 NIU’s respectively and one for packets from the SPU. All the time slots are of equal time interval and equal priority. The complex routing logic takes in the destination coordinates and routes the packet based on the routing algorithm used. After calculating the direction in which the packet must be routed, the routing logic unit will send the control signals to the bus arbiter unit which is responsible for the sending the packet to the respective NIU units. When the packet arrives at the corresponding NIU unit, depending upon the amount of the data in the FIFO, the packet will be sent to the respective node by using AMBRIC REGISTER protocol. The packet size also depends on the various node parameters like ABS register, AIU bus width and Time stamp. As long as the packet filling and packet sending rates are equal in the FIFOs there will be no loss of data.
4.2 Circular FIFO

The circular FIFO is the critical part in this system. The tail pointer of the FIFO increments when any new packet arrives and the head pointer will increment when a existing packet leaves. The relative distance between the head and tail pointers determines the number of packets present in FIFO. The relative distance between the head and the tail pointers is determined by a data count register which increments when a packet enters and decrements when a packet leaves. Once the head or tail pointers reach the maximum FIFO length they will be reset to zero, thus overwriting the stale data. The additional data count register helps in determining whether the FIFO is empty or full, since both the head and tail pointers are the same in this situation. The head pointer specifies the position (count from bottom of the FIFO) of a valid packet in the FIFO for the incoming packet and the tail pointer specifies the position (count from bottom of the FIFO) of the valid packet in the FIFO which is ready to leave the node. The read and
the write cycles are in the same clock domain making the design less complex and thus avoiding additional hardware required to synchronize the two domains.

Algorithm 2 Algorithm for managing the data in a circular FIFO

\[
\begin{align*}
\text{head} & \leftarrow 0 \\
\text{tail} & \leftarrow 0 \\
\text{data\_count} & \leftarrow 0 \\
\text{loop} \\
& \quad \text{if \ } \text{posedge } \text{clk} \quad \text{then} \\
& \quad \quad \text{if } \text{data\_count} > 0 \quad \text{then} \\
& \quad \quad \quad \text{fifo\_empty} \leftarrow 0 \\
& \quad \quad \quad \text{valid\_send} \leftarrow 1 \\
& \quad \quad \quad \text{else} \\
& \quad \quad \quad \quad \text{fifo\_empty} \leftarrow 1 \\
& \quad \quad \quad \quad \text{valid\_send} \leftarrow 0 \\
& \quad \quad \text{end if} \\
& \quad \quad \text{if } \text{valid\_receive} = 1 \quad \text{then} \\
& \quad \quad \quad \text{fifo}\left[\text{data\_count}\right] \leftarrow \text{packet} \\
& \quad \quad \quad \text{data\_count} \leftarrow \text{data\_count} + 1 \\
& \quad \quad \quad \text{tail} \leftarrow \text{tail} + 1 \\
& \quad \quad \text{end if} \\
& \quad \quad \text{if } (\text{accept\_send} = 1) \quad \text{and } (\text{fifo\_empty} \neq 1) \quad \text{then} \\
& \quad \quad \quad \text{data\_count} \leftarrow \text{data\_count} - 1 \\
& \quad \quad \quad \text{head} \leftarrow \text{head} + 1 \\
& \quad \quad \text{end if} \\
& \quad \quad \text{if } \text{head} > \text{fifo\_size} \quad \text{or } \text{tail} > \text{fifo\_size} \quad \text{then} \\
& \quad \quad \quad \text{head} \leftarrow 0 \\
& \quad \quad \quad \text{tail} \leftarrow 0 \\
& \quad \quad \text{end if} \\
& \text{end if} \\
\text{end loop}
\end{align*}
\]

4.3 Network Interface Unit

NIU’s are the means by which each node communicates with its adjacent nodes. Since all the FIFOs here are used for communicating between the adjacent nodes
and the SPU, the communication logic is embedded within the FIFO. This helps in modular reuse of the HDL code for building the NIU. The communication between the FIFOs in two different nodes is based upon the AMBRIC REGISTER model. A total of four NIU’s are used for routing the packets in and out of a node in all four directions. Even the SPU is a type of NIU, but with a interface to the routing table. Depicted in the fig 4.3 are the interconnections between the adjacent nodes.

![Figure 4.3: A interconnection layout between the adjacent NIU’s](image)

The axis is the same as that of the fig 3.1, Node\((x,y)\) is the node of our interest, \(x\) and \(y\) are the coordinates of the node on the DFN grid. The incoming and the outgoing transactions can be out of sync as there are two different FIFOs for incoming and outgoing packets in a NIU. This is the advantage of using different FIFOs in a NIU rather than using a single large FIFO. This also has a advantage
of easily synthesizable logic as there is no need to keep track of overwriting into output and input data spaces. This NIU design results in less congested routing since it needs additional signal to determine if it is a incoming packet or outgoing packet. Apart from this, there will be a additional logic to resolve the conflict between packets when both the incoming and outgoing transactions happen at the same time and for handling the handshaking signals. Since using a single FIFO in a NIU requires additional logic, this approach is slower when compared to using two FIFOs in a NIU. The following two figures show these scenarios.

Figure 4.4: Schematic of communication between two Network Interface Units (NIU)
Figure 4.5: Schematic of communication between two Network Interface Units (NIU)
4.4 Routing Table

The routing table is responsible for routing and for assigning an address to a spike generated on the APP pin. In this design, the mapping is only one-to-one but in reality there can exist a one-to-many mapping for neurons with fanout. This design also cannot handle a single packet going to multiple destinations. In the case of a one-to-many mapping there must be additional hardware to handle more than one spike from the APP since the SPU has a single input data port. Another possible solution for this would be to make the SPU multi ported (with the number of ports equal to the maximum number of spikes that can be simultaneously generated by the ASP) and incorporate the additional logic inside the SPU.

The routing tables can be implemented in hardware by using various kinds of memory like - SRAM, off chip DRAM and BRAM. A trade off needs to be made between the speed and size while considering different types of RAMs or other data storage devices for storing the routing tables. The routing tables can be loaded into the FPGA by a soft-core Microprocessor like MicroBlaze [17] which can in turn interact with the host PC.

![Figure 4.6: Layout of the routing table](image-url)
4.5 SPU (Spike Processing Unit)

The SPU is a extension of the NIU, apart from FIFOs the SPU includes a interface to routing table, logic to map pin numbers with the destination address and packet generation logic. Furthermore the size of the FIFOs in the SPU can be larger when compared to the NIU’s, depending on various factors like - spike generation rate of the APP, pin interface to the APP and spike transfer rate to the APP from the DFN node.

Using the time-stamp on the packet can be helpful in identifying the packet uniquely. This can be useful when simulating the design and in real-time debug of the design. Since the timestamp is just a counter, for larger networks and high packet rates the timestamp bit width must be increased to generate unique timestamps for the packets else there can exist packets with same timestamps in the network at a given instant. In real-time debugging the time-stamp is simply the output of a free running counter. Once the system is bug free, the time-stamp can be turned off as the system does not need to identify each packet, the system only needs to route the spikes to the destination.

4.6 AIU

![Schematic of the AIU used in this design](image)

Figure 4.7: Schematic of the AIU used in this design
The AIU is a large binary encoder which generates the pin number when an APP output pin is set to high. This design uses a simple AIU architecture as shown in fig 4.7 as only a single pin is set to high at a given time. Even though the data processing rate at the interface is faster the communicating with the ASP at a high frequency becomes a bottleneck for the performance.

When multiple pins are excited by the APP there are two possible approaches for encoding - taking a snapshot of the excitations at a uniform interval and processing them (or) a multiplexer can be used at each pin with select input as the APP pin and one of the data lines being the pin number. The first approach occupies less area and has less routing involved in the design but is slower, in the worst case it needs $n$ clocks to scan each block in the queue and generate their respective pin numbers. This approach can be made faster by using a faster clock for scanning and generating the pin number.

Figure 4.8: Hardware implementation of the snapshot approach
The second approach is faster when compared to the first approach, but it needs more hardware. However, using Xilinx CORE IP’s for memory generation can reduce the routing congestion to some extent. However if the pin interface is very large, a BRAM can be used to store the pin numbers and the valid bits register can be used to check if the data is valid, this will reduce the area required to route the logic. The generated data can be loaded into a BRAM in burst mode. Depending upon the output data size of the BRAM, a FIFO can be used as temporary storage while loading the data into the BRAM. While loading the
data into the BRAM it can be checked for valid bits and unwanted data can be removed from the generated data. In simulation a parallel data bus between the NIU’s was used but in hardware Manchester encoding was used to communicate between the NIU’s.

4.6.1 Serial Implementation

![Diagram of serial implementation]

Figure 4.10: Hardware implementation of the serial approach

Serial implementation is very slow when compared to the parallel implementation in receiving the spikes from the ASP. The ASP has to wait for additional clock cycles to transmit the spikes that were generated in a given time slot. Hence there is a trade off between the spike transfer rate and the general purpose IO (GPIO) pin count of the FPGA. This implementation is best suited for smaller FPGA’s or other FPGA boards which support multiple additional features and have fewer GPIO pins. This serial design can be replaced by a design using Rocket IO [2] for
high speed serial communication [5]. Rocket IO is a Xilinx IP core that can be generated by using Xilinx CORE Generator(Coregen) tool.

4.7 AER Packet

The packets use a special representation called the Address Event Representation (AER) [15] [13] [8]. AER is a event driven asynchronous communication protocol in which the sender asynchronously generates on a bus. These events can be merged with other events and can be broadcasted to multiple receivers. In my design the packets containing the spike destination information are generated at the time the spike will arrive and are transmitted asynchronously to the destination node. Depicted below is the structure of a packet.

<table>
<thead>
<tr>
<th>X-coordinate</th>
<th>Y-coordinate</th>
<th>Timestamp</th>
</tr>
</thead>
</table>

Figure 4.11: Packet structure

4.8 Microarchitecture of the Router

The microarchitecture of the router described above is the direct hardware implementation of the algorithm described in the routing chapter.

The left half of the circuit is the mirror image of the right half of the circuit. The left half of the circuit routes the packet along $x$ direction and the right half of the circuit routes the packet along the $y$ direction.

The comparators in the circuit are used to compare the magnitude of the input variables and the output of the comparator is 1 (true) if the equation inside the box is satisfied else 0 (false) if its not satisfied. The ALU units in the circuit perform
Figure 4.12: A schematic of microarchitecture of the Router (DFN)

the basic additions and subtraction of the input variables. It should be noted that the right side input variable is subtracted from the left side input variable. Not obeying this rule may generate unnecessary sign bits resulting in wrong outputs. The register $\Delta x$ stores the euclidean distance along the $x$ direction and $\Delta x_{reverse}$ stores the distance along the torus path. The right half of the circuit performs the same operation as mentioned above but along $y$ direction.

The central portion of the circuit with two comparators and the random bit generator help in choosing the $x$ direction or the $y$ direction. As described previously, the router will route the packet along the direction in which the calculated euclidean distance is less and chooses randomly between $x$ or $y$ direction’s when the euclidean distances along $x$ and $y$ directions are equal. This randomness at the
hardware level is created by the random bit generator block. The random bit generator block is a simple LFSR.

4.8.1 Routing transaction

![Diagram for transaction between router and SPU](image1)

Figure 4.13: Diagram for transaction between router and SPU

![Diagram for transaction between router and SPU](image2)

Figure 4.14: Diagram for transaction between router and SPU
Since it takes one clock cycle to get back the valid signal from a NIU or SPU, once the accept signal is asserted the valid data is accepted by the router in the succeeding time slot.

Depicted in the above two figures are the snapshots of the transaction between the router and the SPU with a time frame of one clock cycle. When the time slot is given to the SPU (fig 4.3) the accept signal to the SPU is asserted and the valid data from the west NIU is accepted by the router. In the second time slot (fig 4.4) the accept signal to the north NIU is asserted and the data from the SPU is accepted by the router.
Chapter 5

Hardware Implementation

The Digital Fabric consists of a large number of modules with asymmetric interconnect, but it is complex to replicate the modules and interconnect them using the “generate” statement in Verilog. It would be easier to generate the Verilog code for the design using a scripting language or a programming language. For the designs presented here C programming language has been used to generate the Verilog code and also for scaling the design. This procedure enables scaling and parametrization of the design on the fly. Eclipse CDT on windows has been used to maintain the repositories and debug a C program for generating Verilog multi-module simulations. This C program is called Autogen.

The Autogen program accepts many command line arguments using these arguments the user can scale the design accordingly. Eclipse IDE provides a user friendly GUI (graphical user interface) to compile a C program and generate the executable files. The Autogen program consists a group of standalone programs which work together to produce the entire Verilog design.

All the standalone executables are combined into a single program using a batch script called auto_gen.bat. The auto_gen.bat accepts all the arguments required by each and every standalone program. The pin interconnects in the routing tables are generated by using LFSR’s (Linear feedback shift register). The routing tables are also generated by the Autogen program. The rand_gen.exe generates the data required by the routing tables, the output of the rand_gen.exe file is the packet_file.dat which is read by the dfn_gen.exe for initializing the DFN node
modules in the top_level.v which is the top level module of the entire design.

C:\> auto_gen.bat <number of nodes along x> <number of nodes along y> <no of APP pins> <width of the ABS reg> <FIFO depth> <detailed log>

Using the top_level.v the design can be further scaled for that particular DFN fabric. The following snippet does the scaling for the generated design.

```
parameter AIU_BUS_WIDTH = 6; //width of the bus between AIU and SPU
parameter APP_PIN_CNT = 1<<AIU_BUS_WIDTH; //pins of the APP
parameter ABS_REG_WIDTH = 4; //size of the x and y coordinates
ifdef SIMULATION
parameter TIMESTAMP = 8; //size of the time-stamp
parameter BUS_WIDTH = AIU_BUS_WIDTH + (2*ABS_REG_WIDTH) + TIMESTAMP; //Data bus between two different NIU's on two different nodes
else
parameter BUS_WIDTH = AIU_BUS_WIDTH + (2*ABS_REG_WIDTH); //Data bus between two different NIU's on two different nodes
endif
parameter FIFO DEPTH = 15; //Maximum number of packets that a FIFO can hold
parameter MAX_X = 10; //maximum number of nodes along x direction
parameter MAX_Y = 10; //maximum number of nodes along y direction
parameter PACKET_CNT = 0; //number of packets sent per clock
```

Depicted in fig 5.1 is the design flow for generating the Verilog file. The figure also shows which standalone executable files are responsible for generating the respective Verilog files for the design.
Figure 5.1: Block diagram of the design flow

```
del /Q HDL
rmdir HDL
mkdir HDL
rm packet_file.dat
rand_file_gen.exe %1 %2 %3
dfn_gen.exe %1 %2 %3 %4 %5 > HDL/top_level.v
AIU.exe %3 > HDL/AIU.v
rt_mod_version.exe %3 > HDL/Routing_table.v
SPU.exe %3 %5 > HDL/SPU.v
self_test.exe %1 %2 %5 %6 > HDL/self_test.v
FIFO.exe %5 > HDL/fifo.v
NIU.exe %5 > HDL/NIU.v
Node.exe %3 %5 > HDL/DFN.v
RNIU.exe %5 > HDL/RNIU.v
```

The above box displays the contents in the `auto_gen.bat`. 
5.1 MicroBlaze and its applications

The MicroBlaze processor communicates with the BRAM using the processor local bus (PLB). MicroBlaze also has other interfaces such as - Ethernet, UART, etc.. Using these interfaces, a communication link can be created between the microblaze processor and the host PC. The petalinux application allows us to load a file from the host PC into the BRAM, this has been described in section section 5.3. Apart from loading routing tables MicroBlaze can be used for three more important tasks. They are controlling the system, system initialization and debugging.

5.1.1 Controlling the system

In the final system several FPGA boards will be used and it will be difficult to control the system as a whole. To solve this problem a MicroBlaze soft-core can be implemented in each and every node and since it has the capability to communicate to the host PC through a Ethernet interface, all the nodes can be connected to a wireless Internet router. All the nodes can now communicate with the host PC via a ssh (secure shell) session. This not only keeps the system clean and simple but a better interface to the user and also provides a number of other features.

5.1.2 System initialization

It is difficult to reset the whole system with a push button interface. Doing a reset by using software is less complex than doing it by using hardware. Since the Atlys board has fewer output ports and almost all the ports are used for communicating with the neighboring boards, an extra port cannot be used for a global reset. To solve this problem all the boards can be put to reset for a certain amount of time
in software by using timers in the MicroBlaze. The packets may be dropped in the FIFO’s if all the boards cannot come back from the reset simultaneously but it is fine as long as the difference between the resets is within 4 clock cycles because a packet takes a minimum time of 4 clocks to reach the neighboring node.

5.1.3 Debugging

Debugging a large system like this one may be tedious and using a Chipscope core (Xilinx IP core to trace the signals) may not be a better idea because this core can take up a large area on the FPGA and also output pins may not be available for probing the signals. To overcome this problem a technique was developed using MicroBlaze to capture the required data and transmit those data to the host PC. This will save resources on the FPGA and this data can be fed to a Matlab program which will analyze the network statistics and track the packets of your interest. More about this program is discussed in Chapter 6.
5.2 PetaLinux and Xilinx tools

Soft-core processors like MicroBlaze and PicoBlaze provide a better interface to communicate with the FPGA by providing on chip programmable connectivity to the hardware logic. The entire design can be created by using the Xilinx tools (ISE and EDK). This design uses the MicroBlaze soft-core processor which has a number of features including and Ethernet port, USB and serial interface. The MicroBlaze executes the PetaLinux [11] RTOS which uses Linux BusyBox (unix utilities).

5.2.1 Compiling the kernel

![Figure 5.2: Configuring the PetaLinux Kernel](image)

Figure 5.2: Configuring the PetaLinux Kernel
The PetaLinux kernel for MicroBlaze can be configured to include various features. Depicted above is a screen-shot for configuring the kernel.

5.2.2 Integrating the MicroBlaze core into the design

Figure 5.3: Integrating the embedded processor with the design in Xilinx ISE

Depicted above is a screen-shot of Xilinx ISE with the top level module and the embedded micro-controller integrated into it. The embedded processor has been designed using the Xilinx EDK suite.

For more details on this section refer Appendix B and C.

5.3 Loading the routing tables

To load the tables into the FPGA a better interface to the host PC is necessary. A driver has been written in C (refer Appendix B.6) to load the tables from a file; the file can be transferred to the design using FTP. The below commands show
how to load the file into the system and load the tables.

```
# dropbearkey -t rsa -f /etc/dropbear/dropbear_rsa_hostkey
# dropbear
```

The above command will create the ssh key and start the dropbear ssh daemon. Now the system is ready to accept ssh connections. The following command will transfer the file to Atlys board using the FTP protocol. The uploaded file can be found in the /var/ftp

```
curl -T work/data.txt -u root:root ftp://192.168.0.2
```

After uploading the file to system the data can be written into the BRAM using a PetaLinux software app from which the routing data will be read by the system.

![Figure 5.4: A screen-shot of the PetaLinux app displaying the content in the file](image)

Figure 5.4: A screen-shot of the PetaLinux app displaying the content in the file
Chapter 6

Packet Trace Capabilities

6.1 Debug, Performance

This chapter describes the verification and debug techniques using the tools that were developed for the design in real-time and simulation environment. Creating a design this large is tedious enough, but debugging it is an even greater challenge. The opportunities for bugs to arise grow exponentially with size, since there are so many more combinations that could go wrong in a design like this.

6.1.1 Debugging in simulated environment

![Process flow for simulated design](image)

Figure 6.1: Process flow for simulated design
6.1.2 Verilog Simulation

After generating the Verilog source code for the design, simulation was performed using ModelSim [9] by Mentor Graphics. In order to enable the simulation mode the following piece of code must be specified in the top_level.v file. However, by default the generated Verilog source code is for simulation mode.

```verilog
define SIMULATION

The parameters in the top_level.v are generated by the Autogen program. Shown below are the parameters which control the design.

```verilog
parameter AIU_BUS_WIDTH = 6; //width of the bus between AIU and SPU
parameter APP_PIN_CNT = 1<<AIU_BUS_WIDTH; //pins of the APP
parameter ABS_REG_WIDTH = 4; //size of the x and y coordinates
'ifdef SIMULATION
parameter TIMESTAMP = 8; //size of the time-stamp
parameter BUS_WIDTH = AIU_BUS_WIDTH + (2*ABS_REG_WIDTH) + TIMESTAMP;
//Data bus between two different NIU’s on two different nodes
'else
parameter BUS_WIDTH = AIU_BUS_WIDTH + (2*ABS_REG_WIDTH); //Data bus between two different NIU’s on two different nodes
'endif
parameter FIFO_DEPTH = 15; //Maximum number of packets that a FIFO can hold
parameter MAX_X = 4; //maximum number of nodes along x direction
parameter MAX_Y = 4; //maximum number of nodes along y direction
parameter PACKET_CNT = 0; //number of packets sent per clock
```

Executing the design in this mode will generate a huge log file which will contain the data present inside the FIFO on each rising edge of the clock. Shown below is the syntax of the log file.

```
APP_\{x\}_\{y\}(0),
data_block_{in_out}_{dir}_{x}_{y}_{bn}( 342801),
fifo_sim_head_{in_out}_{dir}_{x}_{y}(5),
fifo_sim_tail_{in_out}_{dir}_{x}_{y}(5),
X_sim_send_{x}_{y}(1),
Y_sim_send_{x}_{y}(0),
```
The values inside the \{\} are generated when the design is simulated. \{x\} and \{y\} represent the coordinates of the node in the DFN network and \{bn\} represents the block number inside each FIFO block. All the APP-\{x\}-\{y\}\{0\} indicate if there is an incoming spike to the APP located at (x,y) on the DFN network, if there is a spike then the value inside the round brackets will be 1 else 0.

The data_block-\{in\_out\}_-\{dir\}_-\{x\}_-\{y\}_-\{bn\}\{342801\} syntax gives the data present inside the FIFO of a NIU, since there are two FIFO’s (incoming and outgoing) inside each each NIU the \{in\_out\} specifies about the incoming and outgoing FIFO’s, it is represented by 'in’ and 'out' respectively for incoming and outgoing FIFO’s. The \{dir\} specifies the direction of the NIU, there are a total of 6 possible values for this parameter including the SPU - north, east, west and south, aiu_to_router and router_to_aiu. The first four represent the 4 directions and the remaining notify about the outgoing and incoming packets of the SPU respectively. The data inside the round brackets is the packet present in the \{bn\} block of the FIFO.

The X_sim_send-\{x\}_-\{y\}\{1\} and Y_sim_send-\{x\}_-\{y\}\{0\} will keep track of the x and y directions of each and every packet passing through the node. Keeping tracking of the x, y directions of the packets will help in plotting packet path. If a packet travels along the x-direction after traveling through the router the value inside the round brackets will be '1' else '0', similar representation is given for Y_sim_send-\{x\}_-\{y\}.

The fifo_sim_head-\{in\_out\}_-\{dir\}_-\{x\}_-\{y\}\{5\} and fifo_sim_tail-\{in\_out\}_-\{dir\}_-\{x\}_-\{y\}\{5\} represent the head and the tail pointers of the FIFO and the remaining parameters in the \{\} are same as that of the data block. The comma on the trailing end of each line is used for readability.

A part of packet trace will be generated for each and every node on the DFN.
network on every rising edge of the clock. The following table gives an estimate of the data log that will be gathered for a \textbf{10x10} design with a FIFO depth of 15 blocks and a block size equal to packet size (22 bits) for each clock cycle.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Per FIFO</th>
<th>Per NIU/SPU</th>
<th>Per Node</th>
<th>For entire Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP spike track</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Data blocks</td>
<td>15</td>
<td>30</td>
<td>150</td>
<td>15000</td>
</tr>
<tr>
<td>Head pointer</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>Tail pointer</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>Send X</td>
<td>15</td>
<td>30</td>
<td>150</td>
<td>15000</td>
</tr>
<tr>
<td>Send Y</td>
<td>15</td>
<td>30</td>
<td>150</td>
<td>15000</td>
</tr>
</tbody>
</table>

Table 6.1: Estimate of data gathered for a 10x10 design

Accumulating these data will help in debugging the system as well as in tracking every packet in the system for performance analysis. The following sections will describe how these data are interpreted by the scripts for generating the statistics for the design.

In the above figure each frame contains all the data from all of the nodes as described in the table 6.1. Number of frames present in the log file are equal to the number of clock cycles for which the simulation was running.

### 6.1.3 Packing the data into a Matlab array data structure using Perl

Perl is a good scripting language for string manipulation and text pattern recognition. Depicted below is the flow chart for generating Matlab array’s using the Perl script (Matgen).

A frame precisely is a collection of data as mentioned in the table 6.1 for a clock period.

\[i \in \{1,2,3,...,\text{Max(x-dimension of the DFN network)}\}\]

\[j \in \{1,2,3,...,\text{Max(y-dimension of the DFN network)}\}\]
$\text{dir} \in \{\text{north, south, east, west, aiu_to_router, router_to_aiu}\}$

$\text{BlockNumber} \in \{1, 2, 3, \ldots, \text{FIFO Depth}\}$

$\text{inout} \in \{0, 1\}$

The below block of code shows the Matlab array data structure generated by Matgen. Matgen is a program written in PERL which reads the log file generated from the simulation and stores the data into Matlab array data structures.

```
APP(:,:,2,1)=[-1,0,0,0,...];
DFN_X_SEND(:,:,2,1)=[-1,0,0,1,...];
DFN_Y_SEND(:,:,2,1)=[-1,0,0,0,...];
DFN_dir_e(:,:,2,1)=[-1,0,0,0,0,0,0,0,0,0,0,1,...];
DFN_dir_s(:,:,2,1)=[-1,0,0,0,0,0,0,0,0,0,0,0,...];
DFN_packet_track(:,:,2,1) = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,836873,836873,836873,...];
DFN_fifo(:,:,2,1,1,1) = [-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...];
DFN_NIU(:,:,2,1,1,1) = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,279559,279559,1075720,1075720,...];
```
Figure 6.3: Algorithm of the Matgen program
6.1.4 Data analysis using Matlab

Once the Matlab arrays are generated the packets can be tracked using a Matlab script; the script tracks the packets from the source node to the destination node. The packets always follow the pattern shown in fig 6.4 during the transit from source to destination. If the source and the destination nodes are adjacent to each other the intermediate nodes in the path can be skipped from the fig 6.4. The script to track the packets can be divided into 3 parts.

1. This part of the Matlab code will trace the packet until the packet has departed the source node.

2. Since the intermediate path can be of undefined length, the code to trace the packets through the nodes can be written inside a loop and the loop control statement will check for the number of intermediate nodes.

3. This part of the Matlab code is merged with part 2 of the Matlab code which helped to exit the loop when the destination is reached.

Figure 6.4: Pattern for automating the packet trace
6.1.5 Matlab packet tracking algorithm

The following section explains the pseudocode of the Matlab code used for tracking the packets.

Figure 6.5: Packet tracking using Matlab script

Figure 6.6: Screen-shot of packet tracking in Matlab showing a packet’s route from (4,4) to (2,3)
**Algorithm 3** Algorithm for tracking the packets and plotting the path

1. `packet_hops ← MaxHOPS`
2. `trace_packet ← Specify the packet for tracking`
3. `x ← X-coordinate of the source node`
4. `y ← Y-coordinate of the source node`
5. `clock_instance ← 0`
6. `{Get the clock value for which the packet is sent out from SIU}`
7. `while trace_packet ≠ DFN_packet_track(1, clock_instance, x, y) do`
8. 8. `clock_instance ← clock_instance + 1`
9. `end while`
10. `{Get the tail pointer value for the SPU outgoing FIFO}`
11. `while trace_packet ≠ DFN_NIU(1, clock_instance, x, y, 5, 1, i) do`
12. 12. `i ← i + 1`
13. `end while`
14. `Plot_and_dir_packets() {Refer Algorithm 5}`
15. `{After the packet is routed in the correct direction it rests in the outgoing FIFO of a NIU. So, the clock instance for which the packet will be out of the FIFO needs to be found and the NIU direction of the adjacent node has to be calculated}`
16. `cur_tail = DFN_fifo(1, clock_instance, x, y, dir_niu, 2, 2)`
17. `while cur_tail ≠ DFN_fifo(1, clock_instance, x, y, dir_niu, 2, 1) do`
18. 18. `clock_instance ← clock_instance + 1`
19. `end while`
20. `if dir_niu = 1 then`
21. 21. `dir_niu ← 2`
22. `else`
23. 23. `if dir_niu = 2 then`
24. 24. `dir_niu ← 1`
25. `else`
26. 26. `if dir_niu = 3 then`
27. 27. `dir_niu ← 4`
28. `else`
29. 29. `if dir_niu = 4 then`
30. 30. `dir_niu ← 3`
31. `end if`
32. `end if`
33. `end if`
34. `end if`
Algorithm 4 Cont...

\(x \leftarrow \text{next}_x\)
\(y \leftarrow \text{next}_y\)

\{This ends the part one of the script and completes the trace through the source node\}

\textbf{for} \ hop\_count = 1 \textbf{to} packet\_hops \textbf{do}

\(\text{cur\_tail} \leftarrow DFN\_fifo(1, \text{clock\_instant}, x, y, \text{dir\_niu}, 1, 2)\)
\{Refer to Appendix A for detailed explanation of the variables\}
\textbf{while} \ DFN\_fifo(1, \text{clock\_instant}, x, y, \text{dir\_niu}, 1, 1) \neq \text{cur\_tail} \textbf{do}

\(\text{clock\_instant} \leftarrow \text{clock\_instant} + 1\)

\textbf{end while}

\textbf{if} \ DFN\_NIU(1, \text{clock\_instant}, x, y, \text{dir\_niu}, 1, \text{cur\_tail}) \neq \text{trace\_packet} \textbf{then}

\(\text{hop\_count} \leftarrow \text{packet\_hops} + 1\) \{Break out of this loop\}

\textbf{end if}

\textbf{Plot\_and\_dir\_packets()\{Refer Algorithm 5\}}

\(\text{cur\_tail} \leftarrow DFN\_fifo(1, \text{clock\_instant}, x, y, \text{dir\_niu}, 2, 2)\)

\textbf{while} \ DFN\_fifo(1, \text{clock\_instant}, x, y, \text{dir\_niu}, 2, 1) \neq \text{cur\_tail} \textbf{do}

\(\text{clock\_instant} \leftarrow \text{clock\_instant} + 1\)

\textbf{end while}

\(x \leftarrow \text{next}_x\)
\(y \leftarrow \text{next}_y\)

\textbf{if} \ dir\_niu = 1 \textbf{then}

\(dir\_niu \leftarrow 2\)

\textbf{else}

\textbf{if} \ dir\_niu = 2 \textbf{then}

\(dir\_niu \leftarrow 1\)

\textbf{else}

\textbf{if} \ dir\_niu = 3 \textbf{then}

\(dir\_niu \leftarrow 4\)

\textbf{else}

\textbf{if} \ dir\_niu = 4 \textbf{then}

\(dir\_niu \leftarrow 3\)

\textbf{end if}

\textbf{end if}

\textbf{end if}

\textbf{end if}

\textbf{end if}

\textbf{end for}
Algorithm 5 Function to Plot and trace the packets

function plot_and_dir_packets (NULL)
if DFN_X_SEND(1, clock_instance, x, y) = 1 then
    send_x ← 1
    send_y ← 0
    if DFN_dir_e(1, clock_instance, x, y) = 1 then
        send_east ← true (or) check for warp around condition and send along west
        Set dir_niu accordingly
    end if
    if DFN_dir_e(1, clock_instance, x, y) = 0 then
        send_west ← true (or) check for warp around condition and send along east
        Set dir_niu accordingly
    end if
Plot the correct trace along X-direction on the grid
end if
if DFN_Y_SEND(1, clock_instance, x, y) = 1 then
    send_x ← 0
    send_y ← 1
    if DFN_dir_s(1, clock_instance, x, y) = 1 then
        send_south ← true (or) check for warp around condition and send along south Set dir_niu accordingly
    end if
    if DFN_dir_s(1, clock_instance, x, y) = 0 then
        send_north ← true (or) check for warp around condition and send along north Set dir_niu accordingly
    end if
Plot the correct trace along Y-direction on the grid
end if
return NULL
end function
6.2 Real-time Debug

The same method as described above can be used to debug in real-time though this was not done for this work. The signals of interest can be made as outputs of the model and can be fed to the soft-core processor. A simple driver written in C can monitor the pins of the processor and latch the data. After capturing the data they can be written to a file or transmitted wirelessly to the host computer where they can be analyzed by Matlab to obtain various network statistics.

Figure 6.7: Top level view for real-time debug

The MicroBlaze processor communicates with the rest of the system as depicted in fig 1.1 and fig 2.1.
6.3 Derivation for predicting the Path Cost

Path cost is the time taken by the packet to reach its destination from the source at a given time. It is calculated in clock cycles.

![Diagram of a packet's path](image)

**Figure 6.8: Cost for a path**

Depicted above is the pattern followed by a packet during its transit from the source to the destination. The values in the block diagram indicate the number of clocks a packet has spent in a block in the worst case, the value is never a zero for a block.

Hops are the number of nodes a packet needs to pass during its transit, this includes source node and destination node. The minimum number of hops in a path is 2, the case where a packet’s source and destination nodes are the same is explained later in this section. In the fig 6.8 number of hops are 3.

Consider the intermediate node the **IN** and **OUT** blocks in a node represent the incoming FIFO (data coming from neighboring nodes) and the outgoing FIFO (data leaving the current node) respectively. The values in the **IN** and the **OUT** blocks are the depths of the FIFO’s. The **RR** block in the figure represents the time spent by the packet inside the router. When a packet is inside the intermediate node it has to pass through the **IN** block, **RR** block and the **OUT** block. For a given DF there can be any number of intermediate nodes. The following equation
can calculate the path cost for the transit through intermediate blocks.

\[ P_{c_{\text{int}}} = (hops - 2) \times (RR + 2 \times \Delta F_p) \]

**RR** - is the average of all the RR blocks in the path of transit at a given instant.

\[ \Delta F_p \] - is the average of all the time spent by the packet in the FIFO’s in the path of transit at a given instant.

**SPU_{src}** represents the time taken at the SPU to convert the spike into a packet

**SPU_{dest}** represents the time taken at the SPU to convert the packet back into a spike

The term \( (RR + 2 \times \Delta F_p) \) indicates that the packet has to pass through the IN, OUT FIFO’s and the router inside a intermediate block. In this case since the intermediate path is considered in the transit the term \( (RR + 2 \times \Delta F_p) \) has to be multiplied by **hops-2** (neglecting the source and destination nodes).

Consider the source node from the *fig 6.8* when a spike is generated by the ASP it takes one clock cycle for a spike to be converted to a packet in the simulation. This time may vary for a real-time system. After departing from the SPU the packet has to pass through a outgoing FIFO and the router. Similarly the packet has to pass through a incoming FIFO, router and the SPU. In this case at the SPU the packet will be converted to a spike and sent back to the ASP. This time is different from the time taken at the SPU for the source node. All these events at the source and the destination can be combined into a single equation as given below.

\[ P_{c_{\text{sd}}} = 2 \times RR + 2 \times \Delta F_p + SPU_{src} + SPU_{dest} \]

Adding **P_{c_{\text{int}}}** and **P_{c_{\text{sd}}}** will give the total path cost of a packet’s transit from the source to the destination node for **hops \geq 2**. After adding and simplifying the equation the total **P_c** is as follows.
\[ P_c = \text{hops} \times (RR + 2\Delta F_p \times (1 - \frac{1}{\text{hops}})) + SPU_{\text{src}} + SPU_{\text{dest}} \]

\( SPU_{\text{src}} \) represents the time taken at the SPU to convert the spike into a packet

\( SPU_{\text{dest}} \) represents the time taken at the SPU to convert the packet back into a spike

Consider the case where a packet’s source and the destination node are the same.

In this case the packet needs to pass through the SPU twice, once when the spike is converted to a packet and again when the packet is converted back to the spike.

Since the packet contents are checked at the router it needs to spend time in the router. This equation gives the path cost for this case \((\text{hops} = 1)\).

\[ P_c = RR + SPU_{\text{src}} + SPU_{\text{dest}} \]

The generalized equation for the path cost is as given below.

\[
P_c = \begin{cases} 
\text{hops} \times (RR + 2\Delta F_p \times (1 - \frac{1}{\text{hops}})) + SPU_{\text{src}} + SPU_{\text{dest}} & \text{if } \text{hops} \geq 2 \\
RR + SPU_{\text{src}} + SPU_{\text{dest}} & \text{if } \text{hops} = 1 
\end{cases}
\]
6.3.1 Path cost for simulation

In the simulation it only take a clock cycle to convert the spike into a packet and a clock cycle to convert the packet back to spike because in simulation real ASP does not exist and the packet arrival is indicated by setting a bit high in a register. The time taken inside the OUT and IN FIFO’s in the SPU (refer section 4.5) are included in the variables SPU_{src} and SPU_{dest} respectively. But the packet has to wait inside the OUT FIFO after it has been converted into a packet. Therefore, SPU_{src} = \Delta F_p + 1, a ’1’ is added to this variable because it takes one clock cycle to convert the spike to a packet. The packet does not have to wait in the IN FIFO for a long time since in the simulation the packet gets converted into a spike in a single clock as soon as the packet appears in the IN FIFO. Therefore, SPU_{dest} = 1. After replacing these variables in the path cost equation (P_c) and simplifying it, the new path cost equation for the simulation, \( P_{c\text{ sim}} \) is as follows.

\[
P_{c\text{ sim}} = \begin{cases} 
  \text{hops} \times (RR + 2\Delta F_p \times (2 - \frac{1}{\text{hops}})) + 2 & \text{if hops} \geq 2 \\
  RR + \Delta F_p + 2 & \text{if hops} = 1
\end{cases}
\]

Consider a torus network as shown in fig 3.3, it can be seen that the maximum distance between any two nodes along the x-axis is width of the grid - 2. Similarly the maximum distance between any two nodes along y-axis is height of the grid - 2.

\[\forall n > 2, \exists \text{hops} \exists, 1 \leq \text{hops} \leq ((X_{\text{max}} + Y_{\text{max}} - 4) + 1)\]

where n is the dimension of the DF, \(X_{\text{max}}\) is the width of the DF and \(Y_{\text{max}}\) is the height of the DF.
Since the convention here assumes even the source node (refer section 6.3) for calculating the hops, '1' has been added in the above equation.

6.3.2 3D plot of $P_{c\text{ sim}}$

![3D plot of $P_{c\text{ sim}}$]

Figure 6.9: Surface plot of $P_{c\text{ sim}}$

Since $P_{c\text{ sim}}$ is a function of 4 variables one variable is assumed to be constant
for a surface plot. In the following graph $RR$ is assumed to be constant, since $RR$ varies from 1 to 5 it would generate 5 different surfaces.

![Table 6.2: Maximum and minimum path costs for different values of RR](image)

The above Matlab script generates the plot for the path cost. With $RR$ (rr in the above script) set to a constant on each iteration all the outputs are stored in a 2D matrix.
matrix(z) as shown in the above script. The **surf()** function then plots the surface plot based on the magnitude of z.

### 6.4 Results

#### 6.4.1 Wave diagrams

A snapshot of the wave diagram between two adjacent Digital Fabric Nodes has been taken, refer Appendix E for the diagram and explanation.

#### 6.4.2 Simulation results

A total of 16 separate runs have been made to gather the network statistics for each packet rate. These statistics were tabulated below. The conversion from simulation log to Matlab arrays took an average of 2 days for each network.

The table 6.3 shows the network statistics for a 4x4 network and the first row indicates the various conditions at which this network was tested. The **packet generation rate** column indicates the packet rate, for example a packet rate of 2 indicates that a packet is sent from the SPU to the router for every 2 clocks. The

<table>
<thead>
<tr>
<th>FIFO Depth</th>
<th>packet generation rate</th>
<th>Packets sent</th>
<th>Packets received</th>
<th>Packets lost</th>
<th>% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>19654</td>
<td>7928</td>
<td>11726</td>
<td>59.66</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>9822</td>
<td>7898</td>
<td>1924</td>
<td>19.59</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>6545</td>
<td>6508</td>
<td>37</td>
<td>0.57</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>4908</td>
<td>4884</td>
<td>24</td>
<td>0.49</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>3268</td>
<td>3256</td>
<td>12</td>
<td>0.37</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>2449</td>
<td>2441</td>
<td>8</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 6.3: Simulation results for different packet rates for a 4x4 network

*table 6.4* is similar to that of *table 6.3* but the network size is 10x10.
<table>
<thead>
<tr>
<th>FIFO Depth</th>
<th>packet generation rate</th>
<th>Packets sent</th>
<th>Packets received</th>
<th>Packets lost</th>
<th>% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>122867</td>
<td>19250</td>
<td>103617</td>
<td>84.33</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>61408</td>
<td>19355</td>
<td>42053</td>
<td>68.48</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>40924</td>
<td>23549</td>
<td>17375</td>
<td>42.46</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>30682</td>
<td>28853</td>
<td>1829</td>
<td>5.96</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>20433</td>
<td>20165</td>
<td>268</td>
<td>1.31</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>15311</td>
<td>15119</td>
<td>192</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Table 6.4: Simulation results for different packet rates for a 10x10 network

![Graph showing network statistics for 10x10 network](image.png)

Figure 6.10: Network statistics for 10x10 network

Depicted above is the graph between the packet rate and the percentage loss in the packets for a 10x10 network. Moving left on the x-axis increases the packet rate, as it can be seen from the plot that the percentage loss does not change beyond a packet rate of 4. This bottleneck on the performance arises due to the
round robin scheduling.

Depicted below is the graph for a 4x4 network. It can be observed from the two graphs that as the FIFO size increases for a 10x10 network, the packet loss is less for a packet rate between 4 and 6 when compared to a 4x4 network. For a packet rate between 1 and 4 the packet loss is less for a 4x4 network because packets need to travel shorter distance when compared to a 10x10 network. This can be proved from fig 6.9, as number of hops are increasing the cost is increasing hence more probability for the packet to be overwritten. Packets are getting overwritten in the FIFO quickly even when the FIFO size is large. Therefore the parameters which control the packet loss are - size of the network, packet rate and size of the FIFO.

Figure 6.11: Network statistics for 4x4 network
6.4.3 Jitter Calculation

The jitter for a 4x4 network was calculated under the following conditions

- Longest path has been selected taking into consideration the torus connectivity of the network - Source_{(4,4)} \rightarrow Destination_{(2,3)}. Number of hops required to reach this destination choosing either path (torus connection or normal route) is 3.

- Approximately 10,000 packets were released into the network.

- The packet structure was modified to contain the source information for the jitter calculation.

The ideal packet time to reach the neighboring node is 2 clock cycles. This requires the packet to be in the best position in the FIFO and also the best round robin slot during the scheduling. Hence to travel 3 nodes it takes 6 clock cycles.

\[ \delta t_{\text{jitter}} = t_{\text{packet arrival time}} - t_{\text{ideal packet arrival time}} \]

<table>
<thead>
<tr>
<th>Packet generation rate</th>
<th>Packet arrival time</th>
<th>Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16.31</td>
<td>10.31</td>
</tr>
<tr>
<td>2</td>
<td>13.98</td>
<td>7.98</td>
</tr>
<tr>
<td>3</td>
<td>10.07</td>
<td>4.07</td>
</tr>
</tbody>
</table>

Table 6.5: Jitter calculations for 3 different packet generation rates for a 4x4 network

From the above table it can be observed that as the packet generation rate increases (number of cycles taken to release a packet into the network) the jitter starts to decrease because there will be fewer number of valid packets inside the FIFO's at
any given instant as the packet generation rate increases, hence less waiting time. It can be observed from the table that to achieve a jitter value close to zero the packet generation rate should be approximately 5 or 6. This value was not tested because the design had to be simulated for a longer time which results in a larger log file. The existing Matgen program stores the intermediate data in PERL array data structures, modifying the program to store this intermediate data into files will solve the problem.

Brainstorming on techniques to calculate jitter for the DF network has led to self tracing technique for a given path. Refer to section 6.5 for details about this technique.

### 6.4.4 Synthesis Report

![Device Utilization Summary (estimated values)](fig 6.12)

Depicted in fig 6.12 is the screen-shot of synthesis report for the router module (single module). All the major warnings have been resolved and the router module has been replaced by synthesizable state machine [3] logic.
6.5 Self tracing

Consider the fig 6.13, the red box indicates the source node and the green box indicates the destination node. In the simulation when a packet arrives at the destination a register (ASP register) is set to '1' indicating that the packet has reached its destination, if no packet arrives at this node the ASP register is cleared. The ASP register will be refreshed with a new value every clock cycle, if a packet has arrived at the destination, the bit will last only for a clock cycle. The value inside this register is monitored and written into a log file every clock instant. The Matgen program then reads this log file and creates a array data structure (ASP_arr) for all these values. The ASP_arr index is the clock instant and the value corresponding to this index is the value written to the register at that clock instant. Such arrays are created for every node in the DF network.

Back tracing the packet from the destination to its source was achieved by applying the algorithm depicted in fig 6.3 in the reverse direction.
6.5.1 Jitter calculation technique

As mention in section 6.4.3 the Source(4,4) → Destination(2,3) path was selected. The ASP_arr for the Destination(2,3) is selected and scanned for ‘1’ in the array. When a ‘1’ is found in the array the packet responsible for setting this bit is selected and verified for the source information (as mentioned in section 6.4.3 the packet now contains the source information), if the source address matches (4,4) the back tracing algorithm is used to trace the packet back to the source. During this back tracing process the time spent by the packet at each node will be calculated and stored into a separate Matlab array data structure. This process was iterated until all the elements in the ASP_arr array were exhausted.

The Matlab code for calculating the jitter for a path only needs the Source(4,4), Destination(2,3) as inputs.

6.5.2 Advantages of using self tracing and back tracing approach

This self tracing technique is not only useful for calculating jitter it also has the following advantages.

- Faster jitter calculation - This bottom-up approach will eliminate the process of tracing the dropped packets because if the packets were traced from the source node to the selected destination node all the packets leaving the source node have to be traced and verified. These packets will also contain packets which will be dropped during their transit to the destination. Thus following this approach will eliminate the case of checking the dropped packets and thus increase the speed of the calculation.
• Self checking technique to check if a valid packet has arrived at the destination - In hardware this is one of the best approach to check for setup, hold violations in dynamic timing analysis and data corruption due to hardware problems.
Chapter 7

Conclusion and Future work

7.1 Conclusion

A simulation environment was developed to simulate and test the SyNAPSE Digital Fabric network. A software framework has been developed to effectively test and gather the network statistics. The HDL code required for the design can be generated for any size of a network with ease using the Autogen program. The router module of the design has been synthesized for the FPGA, the module has been changed significantly to synthesis the design for the SPARTAN-2 FPGA.

7.2 Future work

Synthesizing the whole system was beyond the scope of this research. In addition a better MicroBlaze subsystem has to be integrated with the design for doing system initialization and to gather the network statistics and to communicate with the host.

The design parameters which control the speed of the system are as follows -

- A synthesizable program to generate spikes which can simulate the behaviour of real neurons.
- Synthesizing the entire system.
- A better mechanism has to be developed to interface the ASP with the FPGA.
• Determining the FIFO size to meet the drop rate requirements.

• A better routing algorithm to reduce the packet arrival time - In larger systems there can be cases where a particular path can be used very often and can cause congestion, this may result in high packet drops. To avoid this the routers must intelligently reroute the packets.

• A better router design which can route the packets in all four directions at the same time - Routing the packets in all four directions will reduce the arrival time, hence it will reduces the packet loss rate because there will be less congestion.

• The Matgen program has to be optimized to take advantage of parallel processing and to use the memory efficiently. This can be achieved by using object-oriented PERL program or by converting the Matgen program into C++ program.
References


Appendices
Appendix A

Matlab Variables

A.1  APP(1,clock\_instant,x,y)

This variable keeps track of the spikes received at the destination ASP’s. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.

A.2  DFN\_X\_SEND(1,clock\_instant,x,y)

This variable keeps track of the direction in which the packet is going to be routed at the router. If it is routed along the x-direction at a particular clock instant it is set to 1 else 0. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.

A.3  DFN\_Y\_SEND(1,clock\_instant,x,y)

This variable keeps track of the direction in which the packet is going to be routed at the router. If it is routed along the y-direction at a particular clock instant it is set to 1 else 0. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.
A.4  DFN_dir_e(1,clock_instant,x,y)

This variable keeps track of the direction in which the packet is going to be routed at the router. If it is routed along the east direction at a particular clock instant it is set to 1 else it is set to 0 if it is routed in west direction. This is used to double check with the \texttt{DFN\_X\_SEND}. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.

A.5  DFN_dir_s(1,clock_instant,x,y)

This variable keeps track of the direction in which the packet is going to be routed at the router. If it is routed along the south direction at a particular clock instant it is set to 1 else it is set to 0 if it is routed in north direction. This is used to double check with the \texttt{DFN\_Y\_SEND}. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.

A.6  DFN_packet_track(1,clock_instant,x,y)

This variable keeps track of a packet passing through the router at a given clock instant. So, if we loop through the clock cycles we can find the packet that passed through the routing node. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant.

A.7  DFN_fifo(1,clock_instant,x,y,dir_niu,in_out,head_tail)

This variable keeps track of the tail and head pointers in the FIFO’s. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant, NIU directions, incoming FIFO or outgoing FIFO and head or tail option.
dir_niu

- 1 - selects the north NIU
- 2 - selects the south NIU
- 3 - selects the east NIU
- 4 - selects the west NIU
- 5 - selects the NIU connecting the AIU to the router (AIU → Router)
- 6 - selects the NIU connecting the AIU to the router (Router → AIU)

in_out

- 1 - selects the outgoing FIFO (NIU → external)
- 2 - selects the incoming FIFO (external → NIU)

head_tail

- 1 - selects the head pointer
- 2 - selects the tail pointer

A.8  DFN_NIU(1,clock_instant,x,y,dir_niu,in_out,head_tail)

This variable gives the packet present in a FIFO block at a given clock instant. The inputs to this Matlab array data structure are x,y locations of the DFN and the clock instant, NIU directions, incoming FIFO or outgoing FIFO and head or tail option.
dir_niu

- 1 - selects the north NIU
- 2 - selects the south NIU
- 3 - selects the east NIU
- 4 - selects the west NIU
- 5 - selects the NIU connecting the AIU to the router (AIU → Router)
- 6 - selects the NIU connecting the AIU to the router (Router → AIU)

in_out

- 1 - selects the outgoing FIFO (NIU → external)
- 2 - selects the incoming FIFO (external → NIU)

head_tail

- 1 - selects the head pointer
- 2 - selects the tail pointer
Appendix B

Petalinux

Petalinux is a SDK (software development kit) for MicroBlaze softcore processor. This replaces the traditional Xilinx SDK, but there are ways to integrate Petalinux SDK into Xilinx SDK if a gui is needed.

B.1 Petalinux Environment Setup

After successfully installing petalinux the environment can be setup using the following set of commands.

```
$ cd <path-to-installed-PetaLinux>
$ source settings.sh
```

You can check to see if the work environment is setup properly using

```
$ echo $PETALINUX /home/user/petalinux
```

B.2 Rebuilding the reference design

cd into the petalinux software directory using the below command.

```
$ cd $PETALINUX/software/petalinux-dist
```

The kernel can be configured to incorporate new features or remove any unnecessary features using the following the command.
Figure B.1: Petalinux kernel menu configuration

```
$ make menuconfig
```

Select the "Vendor" sub-menu and then select the vendor of the PetaLinux reference design that is going to be rebuilt e.g., "Xilinx"

Select the "Vendor Products" sub-menu, e.g., "Xilinx Products", and then select the reference design platform that is going to be rebuilt, e.g., Xilinx-SP605-MMU-full-13.1.

Depicted below is a screenshot of the petalinux vendor selection menu.
After selecting the desired Vendor from the menu **exit** and select **yes** to save the configuration. Run make to recompile the software image.

```
$ make
```

When the compilation is done, the generated images will be in these directories.

```
$PETALINUX/software/petalinux-dist/images and /tftpboot
```

Depicted below is the screenshot of the petalinux compilation progress, the image file is by default named as **image** and has extension **.elf**.
B.3 Testing the software image with QEMU

QEMU is an environment that emulates the software image. Using the following command will make the image boot in QEMU environment.

```
$ petalinux-qemu-boot
```

This command by default boots the latest image.

B.4 Testing the image on hardware

Use petalinux-boot-prebuilt to program the FPGA with the reference design pre-built bitstream.
$ petalinux-boot-prebuilt -p <reference design name> -l 1

The `-l 1` option to petalinux-boot-prebuilt signals to do a Level 1 boot, that is, only configure the FPGA. Level 2 is FPGA + u-boot, and Level 3 is FPGA + pre-built Linux image.

The linux image can be downloaded onto the board using the petalinux-jtag-boot

```
$ petalinux-jtag-boot -i $PETALINUX/software/ \n   petalinux-dist/images/image.elf
```

### B.5 Using C-Kermit

C-Kermit is a opensource network and serial communication software. Using C-Kermit it is possible to communicate with the petalinux system over a serial cable.

The default settings for C-Kermit can be stored in a `.kermrc` file. The following lines of code show the content of the `.kermrc` file.

```
set line /dev/ttyS0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
robust
set file type bin
set file name lit
set rec pack 1000
set send pack 1000
set key \127 \8
set key \8 \127
set window 5
```

The baud rate can be adjusted by changing value for the `set speed` in the above snippet. After creating the `.kermrc` file save it in the user directory (`/home/user`).

The connection can now be initiated by using the following command.
$ kermit -c

Shown below is the kermit output of the .kermrc script.

```
Connecting to /dev/ttyS0, speed 115200
Escape character:  Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
```

B.6 C code for reading a file

The following C code is used to read the data from a file. The file can be uploaded by using FTP to the MicroBlaze subsystem. Once the file has been uploaded, the program can be initiated by the user or can run continuously in the background. This code has to be compiled with the Makefile for generating user application provided by Petalogix.

```c
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
    int x;
    char k;
    FILE *fp = fopen("/home/load_tb.txt","rb");
    printf("cmdline args:\n");
    while(argc--)
        printf("%s\n",*argv++);
    for(x=1;x<=15;x++)
        fread(&k,sizeof(char),sizeof(char),fp);
    printf("k= %c,x=%d \n",k,x);
}
```
Appendix C

Xilinx EDK Design suite

C.1 Setting Xilinx environment

Before launching any of the Xilinx applications on linux we need to set the Xilinx environment. This can be done by using the following command. Go to the directory in which settings32.sh (or) settings64.sh exists

```bash
$ cd <Xilinx installation directory>/<Xilinx version>/ISE_DS
```

Then issue the following command

```bash
$ source settings32.sh
```

(or) if the PC is using a 64 bit OS

```bash
$ source settings64.sh
```

C.2 Building a Hardware project for MicroBlaze

The project can be build from scratch using Xilinx EDK but in this I have modified the existing reference design. The following steps show this can be done.

Go to the `user-platforms` directory in the PetaLinux tree

```bash
$ cd $PETALINUX/hardware/user-platforms
```

Copy the reference design to a desired directory
Here **Xilinx-SP605-MMU-full-13.1** is the reference design and **my-hwproject** is the desired directory.

Go into the **my-hwproject**

```
$ cd my-hw-project
```

Launch the Xilinx Platform Studio(XPS) in **GUI** mode using the following command.

```
$ xps system.xmp
```

The softcore processors hardware can be modified in XPS, after doing so save the changes and select **Generate Bitstream** from the **Hardware** tab.

**Export the design to SDK and exit XPS by selecting the Export Hardware**

![XPS screenshot showing the bitstream generation](image)

**Figure C.1**: XPS screenshot showing the bitstream generation

**Design to SDK** under the **Project** tab. Change directory to fs-boot
$ cd fs-boot

Load the FPGA bitstream with bootloader. This can be done using the following command.

$ make init_bram

Send the hardware parameters to the software platform by running the following command.

$ petalinux-copy-autoconfig

Listed below is the output of the petalinux-copy-autoconfig command
$ petalinux-copy-autoconfig
INFO: Using MSS file ./system.mss
INFO: Attempting vendor/platform auto-detect
INFO: Auto-detected platform "PetaLogix/my-hw-project"
INFO: Using generic kernel platform
INFO: Merging platform settings into kernel configuration
Auto-config file successfully updated for platform
"PetaLogix/my-hw-project"
$

If there were any changes in the software image you need to run the **make** command. If the software image needs to be tested it can be done in QEMU environment by using the following command

$ petalinux-qemu-boot

Now open Xilinx ISE by use the following command

$ ise

Add the .xmp file to the project, it is by default named as **system.xmp**. The **system.xmp** can be found in the EDK project folder, in this case it is the **my-hwproject** folder.

Add the HDL template of the **system.xmp** to the top level module. This can be done by following the below steps.

Select the **system.xmp** from the **Sources** window and then select the **View HDL Instantiation Template** from the **Processes** window.
Figure C.3: Softcore processor template generation

Depicted above is the screenshot for generating the HDL template for the softcore processor, copy and paste the template in the top level file and make any required modifications to connect to different modules.

You may connect any ports of the softcore processor to other components in the top level module.

Run Synthesis, Implement the Design and Generate the bitstream. Once the above steps are completed select the **Update Bitstream with Processor Data** from the processes window.
Figure C.4: Screenshot of the bitstream generation and updating the bit file with the processor data in ISE
Appendix D

Downloading the Bit Image to the FPGA and Petalinux server

D.1 Installing the USB drivers

The bit file can be downloaded onto the FPGA using a xilinx tool called iMAPACT. This tool communicates with the xilinx board through a USB cable, so a USB driver needs to be installed on the host PC. The drivers can be downloaded from the Digilent site from the following link


The vizzini driver module can be installed by the following commands.

```
# modprobe usbserial
# insmod ./vizzini.ko
```

Uninstalling the driver can be done using the following commands

```
# rmmod cdc-acm
# rmmod vizzini
# modprobe -r usbserial
```
### D.2 Downloading the bit file to the FPGA using iMPACT

Depicted below is a screenshot to launch the iMPACT tool. After the bit file has been downloaded onto the FPGA the software image (.elf) file can be downloaded using the following command:

```
$ petalinux-jtag-boot -i $PETALINUX/software/petalinux-dist/images/u-boot.elf
```

If the boot up was successful you should be able to login to the system.

![Screenshot of iMPACT tool](image.png)

Figure D.1: A screenshot from xilinx ISE for launching the iMPACT tool

### D.3 Petalinux webserver

Set the ip address for the server using the following command:

```
# ifconfig eth0 192.168.10.10
```

To check if the server is running properly open a web browser and type in the following address:

```
86
```
http://192.168.10.10

You should be able to see the following page.

Figure D.2: A screenshot of petalinux webpage
Appendix E

Wave Diagram

E.1 Wave Diagram

In the following waveform we are going to track the packet arising from node (1,3) to its destination (1,2). The content in the packet is as shown below.

0010 0001 01101010 101001

The first 4 MSB bits represent the y-coordinate and the next 4 bits represent the x-coordinate, the 6 LSB bits represent the timestamp and the remaining 8 bits are generated randomly before storing inside the RAM.

The accept_spu is set to 1 by the router on node (1,3), in response to this signal if the valid signal(v_spu) is high the incoming data(SPI_data_r_in) to the router from the SIU is accepted and the contents in the packet are read for the destination address. After reading the content the packet is routed according to the routing algorithm, in this case the packet needs to be sent north. A valid signal is set high and the data is sent out through the outgoing FIFO of the north NIU of (1,3) node.

Observing the valid signal from the (1,3) north NIU the packet is received into the incoming FIFO south NIU of the node (1,2). In this case the tail pointer value at this instance is 1 since the tail pointer starts counting from 0 so 1 means the data will be written into block 2 (data_block_2) of the FIFO and the tail pointer is incremented to the next value. The packet will reside in the FIFO until the router sends out
Figure E.1: Wave diagram of a transaction between adjacent nodes. The labels surrounded in the red box indicates a track for simulation purpose only.

an accept signal($accept_s$) to the NIU. In this case the $accept_s$ signal is sent in the next clock cycle itself, observing this the packet is received into the router and decoded. Since both the $\Delta x$ and $\Delta y$ are zero the packet is sent to the AIU where it is converted back to spike and sent to the ASP.
Appendix F

Perl Regex

F.1 Perl Regular expressions (Regex)

The log file is converted into a Matlab script by searching and replacing the strings.

F.1.1 Matching a string

$string = m/match_text/;
The above returns true if string $string contains substring match_text, false otherwise.

$string = m/\w match_text/;
The above statement matches only the strings where the match_text appears at the beginning of a string.

$string = m/match_text$/;
The above statement matches only the strings where the match_text appears at the ending of a string.

$string = m/\w match_text$/i;
The above statement exclusively matches only the "match_text" string.

$string = m/\w match_text$/i;
The above statement does a case insensitive search.
F.1.2 Wildcards and Repetitions

. Match any character
\w Match "word" character (alphanumeric plus ",")
\W Match non-word character
\s Match whitespace character
\S Match non-whitespace character
\d Match digit character
\D Match non-digit character
\t Match tab
\n Match newline
\r Match return
\f Match formfeed
\a Match alarm (bell, beep, etc)
\e Match escape
\021 Match octal char (in this case 21 octal)
\xf0 Match hex char (in this case f0 hexadecimal)

Any character, wildcard, or series of characters and/or wildcard can be followed by a repetition.

* Match 0 or more times
+ Match 1 or more times
? Match 1 or 0 times
{n} Match exactly n times
{n,} Match at least n times
{n,m} Match at least n but not more than m times