FPGA BASED MULTI-CORE ARCHITECTURES FOR DEEP LEARNING NETWORKS

Thesis

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Master of Science in Electrical Engineering

By

Hua Chen

Dayton, Ohio

December, 2015
FPGA BASED MULTI-CORE ARCHITECTURES FOR DEEP LEARNING NETWORKS

Name: Chen, Hua

APPROVED BY:

Tarek M Taha, Ph.D.
Advisory Committee Chairman
Associate Professor
Electrical and Computer Engineering

John Weber, Ph.D.
Committee Member
Associate Dean
Electrical and Computer Engineering

Ralph Barrera, Ph.D.
Committee Member
Adjunct Faculty
Electrical and Computer Engineering

John G. Weber, Ph.D.
Associate Dean
School of Engineering

Eddy M. Rojas, Ph.D., M.A., P.E.
Dean, School of Engineering
ABSTRACT

FPGA BASED MULTI-CORE ARCHITECTURES FOR DEEP LEARNING NETWORKS

Name: Chen, Hua
University of Dayton
Advisor: Dr. Tarek M Taha

Deep learning a large scalable network architecture based on neural network. It is currently an extremely active research area in machine learning and pattern recognition society. They have diverse uses including pattern recognition, signal processing, image processing, image compression, classification of remote sensing data, and big data processing. Interest in specialized architectures for accelerating deep learning networks has increased significantly because of their ability to reduce power, increase performance, and allow fault tolerant computing. Specialized neuromorphic architectures could provide high performance at extreme low powers for these applications. This thesis concentrates on the implementation of multi-core neuromorphic network architecture on FPGA. Hardware prototyping of wormhole router unit is developed to control transmission of data packets running through between cores. Router units connect multiple cores into a large scalable network. This network is programmed on a Stratix IV FPGA board. Additionally, a memory initialization system is design inside the core to realize external
network configuration. In this approaching, different applications could be mapped on the network without repeating FPGA compilation. One application called Image Edge Detection is mapped on the network. Finally this network outputs the desired image and demonstrate 3.4x run time efficiency and 3.6x energy-delay efficiency by FPGA implementation.
ACKNOWLEDGEMENTS

I would like to show my greatest gratitude to my advisor, Dr. Tarek M Taha, for his insight and expertise that greatly assisted the research. I would like to express my gratitude to other committee members, Dr. John Weber and Dr. Ralph Barrera for their support and encouragement.

I also want to thank my colleagues Yangjie Qi, Bin Zhang and Raqibul Hasan. It is an amazing experience to working with them.

And last, but not least, thank all the people for their help directly and indirectly to complete the thesis.
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................ iv  
ACKNOWLEDGEMENTS ................................................................................................................ vi  
LIST OF FIGURES .......................................................................................................................... ix  
LIST OF TABLES ............................................................................................................................... xi  
I. INTRODUCTION ..........................................................................................................................1  
II. RELATED WORK .........................................................................................................................4  
   2.1 Deep Learning Network ........................................................................................................ 4  
   2.2 Routing Technology in Network on Chip ........................................................................... 5  
   2.3 FPGA Implement in Deep Learning Network ................................................................... 6  
III. C++ SIMULATOR .......................................................................................................................8  
   3.1 Router Class ......................................................................................................................... 10  
   3.2 Lookup Table Class ............................................................................................................. 11  
   3.3 Test Bench Class ............................................................................................................... 12  
IV. HARDWARE PROTOTYPING ON FPGA ........................................................................14  
   4.1 Core Design ......................................................................................................................... 14  
   4.2 Wormhole Router .............................................................................................................. 17  
V. EXTERNAL INPUT AND INITIALIZATION OF IN-CORE MEMORY ..........................27  
   5.1 External Data Transfer ....................................................................................................... 27  
   5.2 Initialization Blocks Design .............................................................................................. 29
VI. EXPERIMENT AND RESULTS

  6.1 Simulator Results

  6.2 FPGA Results

VII. CONCLUSION

BIBLIOGRAPHY
LIST OF FIGURES

1. Deep learning network with multiply layers and neurons..............................2
2. A multicore neuromorphic processor..............................................................3
3. Design of a single memristor crossbar neuromorphic processing core ............3
4. MNIST database of handwriting......................................................................5
5. Deep learning network architecture ................................................................5
6. Diagram of Router class................................................................................11
7. Member variables and methods in Lookup table class ....................................12
8. Member variables and methods in Test bench class.........................................13
9. Block diagram of digital core .........................................................................16
10. State machine in control unit of receiving packet.........................................16
11. State machine in control unit of sending packet..........................................16
12. XY network routing.....................................................................................17
13. Packet formation...........................................................................................19
14. Routing conflicts............................................................................................21
15. Block diagram of Handshaking protocol.......................................................22
16. Timing diagram of Handshaking protocol.....................................................22
17. Block diagram of router................................................................................23
18. Block diagram of 5x5 crossbar.......................................................................25
19. Block diagram of data transfer between external input and FPGA ...............28
20. Block diagram of memory initialization modules in core ................................................. 29

21. Power of total FPGA and detail parts ................................................................................ 36

22. Image output from FPGA network .................................................................................... 38
# LIST OF TABLES

1. Parameters of C++ simulator .......................................................................................... 9
2. Runtime results of applications between RISC and FPGA ....................................... 30
3. DE2 board vs FPGA simulator cycle count .................................................................. 30
4. RISC core configuration ................................................................................................. 31
5. Network configuration on DE2 board ............................................................................ 32
6. Multicore FPGA resource utilization on DE2 board ................................................. 32
7. Application throughput on DE2 board ........................................................................... 33
8. Application power consumption on DE2 board .......................................................... 33
9. Processing 100 million pixels in the edge detector on DE2 board ............................ 34
10. Processing 1 million pixels in the MNIST classifier on DE2 board ......................... 34
11. Network configuration with 10 cores on GiDEL board ............................................. 35
12. Multicore FPGA resource utilization with 10 cores on GiDEL board ...................... 35
13. Network configuration with 16 cores on GiDEL board ............................................. 35
14. Multicore FPGA resource utilization with 16 cores on GiDEL board ...................... 35
15. FPGA run time and power consumption on GiDEL board ....................................... 36
16. Application throughput on GiDEL board .................................................................... 36
17. Application power consumption on GiDEL board ...................................................... 36
18. Processing 100 million pixels in the edge detector on GiDEL board ...................... 36
19. Application power consumption without I/O ............................................................. 37
20. Processing 100 million pixels in the edge detector without I/O ........................... 37
CHAPTER I
INTRODUCTION

In recent years it has been shown that reliability and power consumption are the main obstacles to the future growth of multi-core computing systems. Though transistor dimensions on the chip are still getting smaller following 'Moore's law', this trend will not continue forever. Transistors are expected to reach their limits of miniaturization fairly soon, resulting in increased loss of stability and reliability. Reliability and power consumption are among the main obstacles to continued performance improvement of future multi-core computing systems.

As shown in Fig. 1. Deep learning network is a powerful learning technology based artificial neural network in machine learning. Artificial neural network is a class of brain inspired algorithms for efficiently processing and classifying data. They have diverse applications in various fields including pattern recognition, signal processing, image processing, image compression, and Big Data processing [1]. Deep learning network expands traditional neural network to a large scalable network, with larger capacity of neurons and hidden layers. As a result, more complicated data processing applications are able to be mapped into deep learning network.
Interest in specialized neuromorphic computing architectures has been increasing recently. A recent study has shown that many applications such as recognition, mining, and synthesis (RMS) can be simulated through neural networks [2]. RMS applications will play a vital role of future computing systems shown by Intel’s research [3]. Esmaeilzadeh et al. [4] show that several key application kernels can be approximated using neural networks. It can be concluded that neural networks can act from specialized applications to general purpose. IBM recently unveiled the TrueNorth architecture, a 4096 core spiking neural network processor [5].

In prior studies, researchers at the University of Dayton [9] have proposed the design of multicore neuromorphic processors where the weights are stored on chip [6, 7, 8]. As shown in Fig. 3, this system consists of processing cores connected through an on-chip routing network, with one router per core. Two versions of this design were examined: a digital version that utilized SRAM for storing synaptic weights within the core (see Fig. 2) and an analog version that utilized a memristor crossbar array to both store the weights and to calculate the neuron outputs (see Fig. 3).
In this thesis, an FPGA implementation of the multicore digital neuromorphic processor is presented. A software simulator is developed to explore the most optimized options for neuromorphic processors and estimate operation performance. The design of the digital neuromorphic system has a significant amount of similarities to the analog memristor neuromorphic system in terms of the control logic in the cores, the input and outputs to the cores, and the routing systems. Dynamic routing and Wormhole switching algorithm are applied in the routing system. External data stream transmission system between external inputs to the FPGA are developed to allow network configuration and memory initialization to make the system reconfigurable.
CHAPTER II
RELATED WORK

Related works in this thesis concentrate on deep learning network, routing technology in network on chip and FPGA implements.

2.1 Deep Learning Network

Neural networks are currently highly popular in many areas such as pattern recognition, digital data analyzing and signal processing. With the growth of human demands and data scaling, deep learning networks are replacing more and more traditional applications in areas such as image processing. According to recent research, Big Data is the most booming data processing application with the rapid growth of digitalization. However many conventional software solutions are not able to manage exponential data size in Big Data. Compared with software, a hardware with new architecture which is match for Big Data could solve the dilemma. Therefore, deep learning network is quickly growing at current stage [10]. A deep learning network uses a deep and large scale neural network and is able to learn feature from big data distributions, such as the MNIST database. This is database of handwritten digits that is commonly used for training various image processing systems (see Fig. 4). A giant size of database is required for better rate of precise handwriting recognition. The structure of neural network could deal with large data matrix easily (see Fig. 5). Moreover, deep
learning network is able to map different data processing applications without changing to different algorithms. Hence deep learning network is match for the needs of most applications such as MNIST database.

![Fig. 4. MNIST database of handwriting](image)

![Fig. 5. Deep learning network architecture](image)

2.2 Routing Technology in Network on Chip

Networks on chip are a key feature of neuromorphic processor. The on-chip routing network allows the cores to communicate with other cores. Additionally they allow external inputs to be routed to any core and outputs from any core to be routed to any external components. These external components need to have a connection to the on-chip routers. In a deep learning network, huge amounts of neurons exist in the network. According to this structure, big numbers of core processors should be integrated as a network in a neuromorphic processor. Routing technology allows to move data streams in a proper path inside the network. In many previous research, the common approaching is static routing which core units are connected with each other in fixed configuration and data streams are transferred to destination core directly from source
core [11]. Although a network based on static routing would be faster and simplify hardware prototyping, it suffers from congestion problems due to unpredictable and time-varying loading from neighbor routers [12]. Furthermore, static routing is not scalable since its chip area will increase rapidly with the number of cores as more routing lines would be needed to connect a larger set of cores.

In order to overcome shortcomings of static networks, a new paradigm of network on chip is developed to replaces the traditional fixed point to point connections with links connected through programmable routers. In this approaching, the path from the source to the destination is determined by routing algorithms in routers. The first significant advantage of dynamic routing network is the reliability and effect of system. This comes from the invariant wire resources cost in each routing unit. Also this design has fault tolerant feature while network scale enlarges. In addition, wire resources could be reusable by packet-based communication methods with increasing core numbers and scale of network. Moreover, dynamic routing network enables high-bandwidth data throughput and flexible network expanding connection for complicated network construction [13]. Therefore, dynamic routing network satisfies the needs of large deep learning network in neuromorphic processor.

2.3 FPGA Implement in Deep Learning Network

From prior research, many neural networks are simulated in software. Much work based on computer simulations has proved the high capability and fault tolerant of deep learning networks to map various systems in engineering, science, economics fields. However, software simulations consumes much run time on computers since data
transmission has low efficiency between CPUs to hard disks or on-board memories. Real-time applications require high speed computations. As a result, software simulators based on deep learning networks on computer can be slow. People need other low-cost and high-speed approaches. Therefore hardware implementations have been taken into consideration by many researchers. Among various methods of hardware implementation, Field-Programmable Gate Array (FPGA) demonstrates the advantage of higher accuracy, better repeatability, lower noise sensitivity, higher repeatability and flexibility, and compatibility with other types of implementation. On the other hand, other implementations such as analog systems are more difficult to be designed. In addition, less flexible is another limitation for scaling network size, or for very specific applications [14]. FPGAs are good solutions for neural network implementations. Firstly, FPGAs preserve the parallel architecture which is similar to neural networks. The most important feature of FPGAs is flexible programmability and reconfiguration by users. FPGAs are an easy method to hardware prototyping and as a way to achieve low cost, flexible hardware designs. The ease of FPGA helps to establish processors which are general purpose.
Before the level of hardware prototype design, software simulator helps to evaluate and determine the optimized options for hardware. In order to explore different core and routing options, it is much easier to and more efficient to use a simulator than to modify the FPGA design. For software programming of simulators, two possible programming languages could be implemented to create simulator: MATLAB and C++. MATLAB is a common software tool designed for high end engineering fields and image graphic processing. It is a powerful toolkit which is specifically designed for complex matrix computations and solving linear equations. But compared with many other common programming languages, MATLAB has several disadvantages. The most severe one is that it is a highly packaged language and therefore it will execute more slowly than most compiled languages. This problem can be mitigated by structuring coding in a proper way, but slow efficiency of MATLAB program still cannot be fully solved when scaling network larger in the simulator.

Object-oriented languages focus on components that the user perceives, with objects as the basic unit. This technology can easily describe the simulator in high-level concepts and abstractions. Also object-oriented languages are flexible and reusable. They keep from re-inventing the same variables and re-writing the same functions for different
situations. This helps to easily explore a large number of different core and routing options. In object-oriented programming, modules in the processor could be developed as classes. Each class defines the kind of data member it contains and any logic sequences that can manipulate it, which are called Methods. These data members could be defined as registers and parameters in RTL hardware. Methods control or proceed data in the class. Methods could be overloaded to establish different approaches. In this method, different core and routing options could be explored by overloading different methods in a class. Table 1 shows all possible parameters in the simulator. The definitions of classes is reusable by program and more easily distributed for a large network.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_CORE</td>
<td>Core amounts established in network</td>
</tr>
<tr>
<td>BITWIDTH_BUS</td>
<td>Bit width of bus wire connected in network</td>
</tr>
<tr>
<td>CYCLES_BODY_PASS</td>
<td>Cycles cost of processor computation</td>
</tr>
<tr>
<td>CYCLES_CUMMNICATION</td>
<td>Cycles cost of communication protocol</td>
</tr>
</tbody>
</table>

In the hardware prototyping, the whole design contains three main modules: core module, router module and test bench module. Considering concentration on evaluation of runtime but not the exact results from simulator. The core module could be simplified to reduce coding time and compilation load by stalling core modules while the simulator runs. The cycle amounts could be estimated by hand easily according to counts of data packets. By this method, core unit is programmed as a method integrated in the router class instead of a class, for boosting the runtime of simulator. Therefore, only router module and test bench module will be programmed as Class in simulator. Test bench
modules and routing unit are variable according to routing methods. Bit-width of data or address and scale of network are defined as parameters to achieve variable configurations easily. Cycle numbers are collected to estimate the runtimes of different core and routing options.

Router network is the main part in the simulator. Wormhole routing network is implemented in this project. The modules of wormhole network are packaged as varieties of Classes: Router, Lookup table and Test bench. Router Class provides methods to transmit data packets and control flows of packets. Lookup table Class provides the initialization methods to set up Routers in the network in order to map different applications. Test bench Class sends out input data packets and retrieve output packets from network.

3.1 Router Class

The class of Router simulates the operation of router in the network. The class receives data flits from other routers or test benches and withhold them when the transfer traffic is blocked. It opens the corresponding crossbar channel according to the coordinate values contained in the head flit, and closes the occupied channel when receiving the tail flit. The cycle numbers which handshaking protocol and core calculation consume could be easily estimated by hand. So the core part is integrated inside of router as a method.
The cycle numbers are accumulated when router class is running by iteration. In addition, the router has a display method to help to monitor the traces of data flits. Fig. 6 shows the whole diagram of Router class.

3.2 Lookup Table Class

In the project, it implements an X-Y on-chip routing network. Each routing units are interconnected with each other as a grid. In the simulator, it makes coding work very intensive and slows down the running speed of programs. Linked data structure technology is applied to resolve this issue. In C++, a linked data structure is a basic data structure where each item contains a pointer which we need to get to the next item like a train. The main advantage of linked data structure is that the links provide us with the capability to rearrange the items efficiently. In the simulator, Router instants are connected in a similar way of the linked data structure. Compared with the fixed routing
unit connection, linked data structure means that next router number varies by changing the mapping methods between different applications. Therefore Lookup table class records the number of next destination router. Fig. 7 demonstrates all member variables and methods in Lookup table class. Each of the Router instant reads its own lookup table to set up next the routing unit connection when initialization the simulator. In addition to number of next router, channels of input and output and amounts of cycles for which router holds data flits are also recorded. By applying linked data structure technology, it is convenient to mapping different applications onto neural networks by just changing lookup tables.

```cpp
class Lookup_Table {
  private:
    int offset;
  public:
    int Num_Packet;  //Number of package
    int Header_Pointer[AMOUNT_POINTER];  //Pointer for depth of table
    int Router_Path[DEPTH_TABLE];  //All routers path
    int Channel_Direct_Input[DEPTH_TABLE];  //Channel of package in input direction
    int Channel_Direct_Output[DEPTH_TABLE];  //Channel of package in output direction
    int Type_Cycle[DEPTH_TABLE];  //Types of cycles for calculation in the 10 cell
    int Data_Amount[DEPTH_TABLE];  //Amounts of datum
    int Enter_Core[DEPTH_TABLE];  //Enter into core
    int HDR[DEPTH_TABLE];  //Head, Body, Tail

    Lookup_Table();

    void Clear();
    void Read_FILE(string filename);
    void Display();
};
```

Fig. 7. Member variables and methods in Lookup table class

### 3.3 Test Bench Class

Test bench class has two functions. First sending data packet in the first layer of neural network, which means emitting data into the network. Second is receiving final result packet form the last layer of neural network. Test bench also owns its lookup table to connect the first following routing unit in the network. Fig. 8 demonstrates all member variables and methods in Test bench class.
Fig. 8. Member variables and methods in Test bench class

```cpp
class Sender_Receiver {
  private:
  IO_Cell IN_IO_Cell_Unit;
  IO_Cell OUT_IO_Cell_Unit;
  IO_Cell *Pointer_IO_Cell_Next[DEPTH_MULTI_SEND];
  Lookup_Table *Pointer_Table;

  int Flit_Count_REC;
  int Flit_Total_REC;
  int Flit_Count_SEND;
  int Pointer_MULTI_SEND;
  int Num_Package_Send[DEPTH_MULTI_SEND];
  int Flit_Total_Send[DEPTH_MULTI_SEND];

  Sender_Receiver();

  void Clear();
  void Send();
  void Receive();
  void Run();
};
```
CHAPTER IV
HARDWARE PROTOTYPING ON FPGA

In this paper an FPGA implementation of the multi-core architecture for deep learning network is presented. The design of the digital neuromorphic system has a significant amount of similarities to the analog memristor neuromorphic system in terms of the control logic in the cores, the input and outputs to the cores, and the routing systems. The key differences are in each core’s neuron compute circuits (memristor based or SRAM/adder/multiplier based). Hence designing the digital design on an FPGA will help determine the peripheral and routing logic for the analog cores as well.

4.1 Core Design

Core unit performs calculation process of data packets which enter into core module. Fig. 9 shows a block diagram of our proposed digital core for processing the feed forward neural network described in equations (1) and (2). Each core processes a collection of N neurons, with each neuron having up to M input axons. $(W_{i,j})$ are stored in a memory array directly within the neural core. When an input pre-synaptic neuron is received, it is multiplied with the pre-synaptic input values $(x_i)$ for all the neurons in parallel and summed into an accumulator (one per neuron). Once the final output neural values are generated, they are copied into an output buffer.
The values in the output buffer are sent to the router associated with the core serially. During the serial transmission, each value in the output buffer passes through an activation function unit that implements equation (2). Additionally, the values are sent through a routing table on the core that provides the destination synapse number associated with the output.

The calculation for all the neurons are carried out in parallel, thus leading to significant speedups. The serial transmission of the outputs is not a problem in this design as the core can only operate on one input synaptic value (this is used by all the neurons in parallel). The use of an output buffer allow the core to overlap data transmission and neuron calculations for a new set of inputs, thus optimizing the system. The intra-core synaptic weight memory also allows the cores to operate at high speeds as this data does not have to be brought in from outside the core. A control unit coordinates the activities of all the components on the core including initialization, processing inputs, neuron computations, and data transmission. Fig. 10 and 11 show the designs of state machines.

\[
v_j = \sum_i W_{i,j} x_i \tag{1}
\]
\[
y_j = f(v_j) \tag{2}
\]
Fig 9. Block diagram of digital core [9]

Fig 10. State machine in control unit of receiving packet

Fig 11. State machine in control unit of sending packet
4.2 Wormhole Router

In terms of network on chip, many network structures could be applied. Usually, XY network is most common structure. XY network shown in Fig. 12 is defined as a dimension order network where data packets move in horizontal direction and then in vertical direction to the destination unit [15]. XY network expends in a mesh topology. Addresses of the routers are their XY-coordinates. The advantage of applying XY network is the simple routing algorithm. It leads to ease design of hardware prototyping and low cost of FPGA hardware cost.

![Fig 12. XY network routing [16]](image)

4.2.1 Routing Technology

The routers in the network implement certain switch technology to transfer the packet into different directions. Switching techniques can be classified based on network characteristics and it contains two basic switch architectures: Circuit switched and packet switched.

Circuit switched network works as a static routing method. It means establishing a fixed physical wire path before transmitting the data packets. The routing unit only works
as building paths from source and destination. Therefore circuit switched network wins high speed of data packet transmission because of none buffering or blocking steps occur during the network. Moreover, it consumes less hardware resources because none of buffers or registers are placed inside the switch units. However circuit switched network is extremely weak in case of establishing a large neural network. Huge width of bus channels will be consumed. Therefore, packet switched technology is introduced to solve the frustration of Circuit switched one.

Packet switched networks work in a dynamic switching method. It means data packets are transmitted by each routing unit without reserving the entire path in advance. Routing units control data flows when data packets enter and pass. The channels in the router are created by routers according to path direction information which is contained in the packet flits. Data packets will go through if the certain channel is open, or block if the channel is closed. Moreover, it takes less hardware resources and easier algorithm to realize multi-casting and receiving functions. The most advantage of packet switched networks is the large scaling capability of network without more hardware resources. In the cost of this flexible characteristics, more cycle time is consumed in order to lead the packets pass through the channels between the routers. However, nowadays the processors share a high clock frequency with less power consumption. Powerful processors are able to weaken the latency happening during the transmitting. Packet switched networks is classified as Wormhole, Store and Forward (S&F), and Virtual Cut-Through Switching (VCT) networks [17]. By comparing the characteristics with these three switching technology, Wormhole switching wins a simple hardware architecture and costs less buffers and registers. This demonstrates that Wormhole switching is match
Wormhole switching is a variant of the virtual cut-through technique that avoids the need for large buffers for saving messages. In wormhole switching, a packet is separated into amounts of data flits which are the smallest units on which flow control can be performed. These flits contains three members: Head, Body and Tail flits, shown in Fig. 13. The head flit carries routing addresses of X-Y coordinates. Also the amount of the following flits are contained in the Head flits. The flit following is named Body flit, which carries the synapses addresses and the input element datum heading to the destination processor. The final flit is Tail flits. It closes the occupied channel which the flit passes through. The main advantage of wormhole switching derives from the pipelined message flow, since transmission latency is insensitive to the distance between the source and the destination. Moreover, since the message moves flit by flit across the network, each node needs to store only one flit. The reduction of buffer requirements at each node has a major effect on the cost and the size of systems.

![Packet formation](Fig. 13. Packet formation)

**4.2.2 Wormhole Routing Algorithm**

The first step in the transmission of data from one core to another is to send a header flit from the source core. This header flit makes its way through a series of router using dimensional routing (first route along the x axis and then along the y axis), establishing connections in each routers crossbar, so that the source core is directly
connected to the destination core. Handshaking is used between the routers to set up this connection.

Once the connection between the source and destination cores is set up, the neuron outputs are transmitted from the source core through the routers to the destination core. Each neuron value is accompanied with a destination synapse number. Once all the synaptic values are transmitted, the source core transmits a tail flit that closes the connection between the source and destination cores in the routers.

Each router contains five input buffers that store only the header or tail flits. These are needed as the round robin arbiter may need several cycles before it can process the header/tail flit. Synaptic output flits are transmitted between the header and tail flits, and these do not get stored in the routing buffers and they do not need service from the router arbiters. This approach allows the amount of buffers to be significantly reduced (enough to store just one header/tail flit), and thus reduce the area and power consumption of the routers.

If two cores try to set up routing channels with overlaps in their path, then one path will be blocked until the other is freed. This is illustrated in Fig. 14, while the established path A blocks path B from being formed. Path B can be formed only after path A is closed. This has the potential to lead to deadlock situations where four paths are trying to form and are all blocking each other. The only solution to this in a dynamic wormhole routing network is to use virtual channels. Given that the connectivity between neurons in an MLP are fixed, the connectivity between cores can be predicted ahead of time. Therefore, we assume that the mapping of the neurons to cores can be carried out to
ensure that deadlocks do not occur.

![Routing conflicts diagram]

Fig. 14. Routing conflicts [9]

### 4.2.3 Handshaking Flow Control

In the network, multiple data packets attempt to use the shared network resources while resources are limited usually. Actions should be taken to control these flows to avoid loss of data when flows have resource competition. That means some data flows must be blocked while others proceed. In this wormhole network, each router has only one or few data buffer in one channel. Head flit will be buffered for several clock cycles in the router before router opens channel according to the destination information in the Head flit. At that time, all the following data flits must be blocked until Head flit is transmitted to other module.

Req/ack handshake protocol technology is applied to control the flows. Fig. 15 and 16 shows how this protocol works. As the name shows, one module send out its “Request” hand when it requests the communication from another, and another sends out its “Acknowledge” hand if it responds the request. Finally the communication is
accomplished through handshake. In the hardware, it applied the similar communication form. The source buffer and destination buffer are connected with “Request” and “Acknowledge” signal wires besides the data wires. The source buffer asserts “request” signal to high when it holds data flit to send. The destination buffer responds “Request” signal and decides whether to send “acknowledge” signal by detecting if destination buffer is full or empty. It assert “Acknowledge” signal to high when empty bit is detected in the buffer. The source buffer starts to emit the data flit only after the “Acknowledge” signal is high.

Fig. 15. Block diagram of Handshaking protocol

Fig. 16. Timing diagram of Handshaking protocol

4.2.4 Router Module Design

Each router contains a collection of five input buffers which is able to store only one Head or Tail flit out of the whole packet. These five input buffers connect with one processor and other four routers around. In the meantime, input buffers are linked to one
of two Crossbar units and an Arbitration unit internally. The Arbitration unit schedules five inputs from the input buffers and sends the result to the Control unit. The Control unit opens or closes the path channels for two Crossbar units. One Crossbar unit switches the Head or Tail flit to the certain direction of the output, while another Crossbar unit switches the Body. The following Modifier unit changes the routing addresses in the Head flit. Fig. 17 shows the block diagram of router.

Fig. 17. Block diagram of router

Input buffers store data flits which enter into router. One router module is connected with five other modules. One is core module and the other four are router modules in four directions: up, left, down and right. Therefore there are five groups of buffers are associated with external data buses from these five modules. Each input buffer could buffer only one data flits to meet the requirement of reduction of hardware cost. Inside input buffer module, it could be divided into two sub module: buffer and control unit. Buffer module is FIFO which consists of registers which store multiple data flits. Buffer module is controlled by Control unit module which applies Handshaking protocol as flow control when a data flit requires to enter router and to leave to next module.
Moore state machine is designed to realize functions of Handshaking protocol and manipulate buffer module to store or send data flits.

Selection module is considered as a switch arbiter to arrange which data flit could run through router when traffic congestion occurs. It is connected with five groups of input buffers. Selection module utilizes Round-robin scheduling to scheme flits sequences when multiple data flits send require signals.

Round-robin scheduling can be used as a conflict controlling approach of first-come first-served queuing. A router which provides round-robin scheduling has a separate queue for every data packet flow, where a data packet flow is identified by its own source and destination address. Round-robin scheduling is an easy scheme approaching without complex adoptive scheduling algorithm. Its algorithm only lets every active data packet in the queue to take its turn in transferring packets in a shared channel by a periodically repeated order. This means the former order of data packet will always be chosen prior than the latter one when two data packets arrive simultaneously. If one flow is out of packets, the next data packet flow will take place. Hence, Round-robin scheduling tries to prevent data flows congestion in an easy approaching.

Decode module performs packet destination decoding as named. According to packet formation of wormhole routing technology, packet destination information is carried by Head flit. Therefore Head flit will enter decode module as long as it buffers into input buffer. Decode module analyzes destination information from Head flit and then transmits information to control unit of crossbar.
Crossbar module directly connects the five direction input ports to five direction output ports of router. In effect, a crossbar shown in Fig. 18 contains five groups of multiplexers. Each group of multiplexer connects five input ports with one output port. Signal of selector input manages each multiplexer to decide connection between only one of five input ports to output port. This architecture would be typical where multiple input ports are connected with multiple output ports in non-blocking way.

![Fig. 18. Block diagram of 5x5 crossbar](image)

Control unit of Crossbar sends selection signals to multiplexers in crossbar module. Inside control unit, five groups of sub controllers are designed for five output directions. Each sub controller receives destination information from decode module. Then it checks occupation status of required channel direction and output corresponsive selection signal to the multiplexer, opening the channel for incoming Body flits towards crossbar unit. If the required output channel is already occupied by previous flits, this flit
will be stalled in the input buffer to avoid data conflicts. The occupied channel will restored by Tail flits.

Address modifier module changes destination information from Head flit before Head flit leaves from the current router. The X and Y coordinate value changes while flits are transmitted up, left, down and right.
CHAPTER V
EXTERNAL INPUT AND INITIALIZATION OF IN-CORE MEMORY

5.1 External Data Transfer

In the prior design, memory array were hardcoded. This means memory array is initialized by reading memory initialization file from hard disk and storing it on the memory array. It is an easy memory initialization approaching which costs less Verilog coding and logic units on FPGA board. But two critical disadvantages of this approaching ruin the aesthetics of whole system. One disadvantage is that contents in memory array are fixed after initialization from reading files from hard disk. The result is that only one application could be mapped on the processor each time. Another disadvantage is that multiple full compilations of Verilog code is required if the memory initialization file is modified in order to mapping different applications on the processor. Performing full compilation of Verilog consumes long time. These disadvantages makes the processor design inflexible for general purpose. External inputs transfer is needed.

In order to allow data stream is transmitted into the processors from off chip, the data bus ports are increased in the previous FPGA design as shown in Fig. 19. Data bus ports include data bus wires which carry data streams, and several bits of control signal wires which control the communications between the FPGA module and external inputs.
The external input part is now implemented in a new larger FPGA board (a GiDEL Prostar IV). This FPGA board provides simple FIFO IP to transfer data from on-board memory to user’s module [18].

In case of writing to FIFO, the write request signal is asserted high when the data is ready from source port data bus. The acknowledge signal on the destination port will set high on the next clock if the destination port accepts. The acknowledge signal will not be set to high if memory in destination module is full. The source port will send out data on each rise edge of clock only if both write request and write acknowledge signals are asserted high.

In case of reading from FIFO, the read request signal is asserted high when source port data bus request reading data. The read acknowledge signal rises to high when the data is ready for reading. The read acknowledge signal will not be set to high when the data is not ready in destination module. The destination port will send out data on each rise edge of clock as long as both read request and read acknowledge signals are asserted high.
5.2 Initialization Blocks Design

The configuration data for in-core memories utilize the data bus mentioned above. It includes memory datum of routing tables, activation function and weight files. Internal memories are established to store these data in each core. Each memory has input and output ports. The output ports send data to the destination module. The input ports connect with the data bus ports in order to receive routing tables, activation function and weight files from external inputs. An arbiter module is developed to dispatch those three memory initialization data from input data bus. It analyze flag bits from data flits and then distribute each flit to its destination memory array. The block diagram shown in Fig. 20.

Fig. 20. Block diagram of memory initialization modules in core
CHAPTER VI
EXPERIMENT AND RESULTS

6.1 Simulator Results

In the simulations, four applications are applied to mapping on the simulator. They are Edge Detection, Deep network, OCR and Object Recognition. The results shown in Table. 2 are FPGA implement on deep learning network consumes less cycles and boosts the runtime of applications. Table. 3 demonstrates similar cycle counts of edge detector between FPGA and software simulator.

### Table. 2. Runtime results of applications between RISC and FPGA

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC Simulator [19]</th>
<th>FPGA Simulator Developed</th>
<th>FPGA Simulator Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.30</td>
<td>2.02</td>
<td>5.17</td>
</tr>
<tr>
<td>(million pixels/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep network</td>
<td>0.08</td>
<td>7.41</td>
<td>86.84</td>
</tr>
<tr>
<td>(million pixels/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCR</td>
<td>0.30</td>
<td>10.11</td>
<td>33.93</td>
</tr>
<tr>
<td>(million pixels/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recognition</td>
<td>0.18</td>
<td>10.04</td>
<td>55.75</td>
</tr>
<tr>
<td>(million pixels/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table. 3. DE2 board vs FPGA simulator cycle count

<table>
<thead>
<tr>
<th>Application</th>
<th>DE2 Board</th>
<th>FPGA Simulator Developed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>131</td>
<td>139</td>
</tr>
<tr>
<td>(cycle count)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2 FPGA Results

6.2.1 DE2 FPGA Results

The multicore neuromorphic processor was implemented on an Altera DE2-115 board. This contains an Altera Cyclone IV FPGA (part EP4CE115). The system was programmed in Verilog and compiled using Quartus II. The number of neurons per cores, bits per neuron, and synapses per neuron were made compile time variables so that different design options could be examined. The synaptic memory was implemented using the onboard memory within the FPGA. Two applications were implemented on the FPGA based neuromorphic processors: edge detection and MNIST. To evaluate the performance of the FPGA, the execution of the applications on a RISC processor [22] is simulated. The most efficient version of the application was simulated on the RISC processor for a fair comparison. The RISC system utilized modeled an ARM processor with the configuration shown in Table 4.

Table. 4. RISC core configuration

<table>
<thead>
<tr>
<th>Configuration Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>1 Ghz</td>
</tr>
<tr>
<td>Frequency</td>
<td>1 Ghz</td>
</tr>
<tr>
<td>Functional Units</td>
<td>1 Integer ALUs</td>
</tr>
<tr>
<td></td>
<td>1 Integer multiplier/divider</td>
</tr>
<tr>
<td>Fetch/Decode/Issue/Commit width</td>
<td>1/1/1/1</td>
</tr>
<tr>
<td>Fetch Queue Size</td>
<td>4</td>
</tr>
<tr>
<td><strong>Branch Logic</strong></td>
<td>bimodal, two-level 1 KB table</td>
</tr>
<tr>
<td></td>
<td>8 bit history</td>
</tr>
<tr>
<td>Predictor</td>
<td>bimodal, two-level 1 KB table</td>
</tr>
<tr>
<td></td>
<td>8 bit history</td>
</tr>
<tr>
<td>BTB Size</td>
<td>1 KB</td>
</tr>
<tr>
<td>Miss-prediction Penalty</td>
<td>3 cycles</td>
</tr>
<tr>
<td><strong>Cache and Memory</strong></td>
<td>16 KB, 4-way, 64 byte blocks</td>
</tr>
<tr>
<td></td>
<td>2 cycle latency</td>
</tr>
<tr>
<td>L1 Instruction Cache</td>
<td>16 KB, 4-way, 64 byte blocks</td>
</tr>
<tr>
<td></td>
<td>2 cycle latency</td>
</tr>
<tr>
<td>L1 data Cache</td>
<td>16 KB, 4-way, 64 byte blocks</td>
</tr>
<tr>
<td></td>
<td>2 cycle latency</td>
</tr>
<tr>
<td>Main Memory</td>
<td>200 cycle latency</td>
</tr>
</tbody>
</table>

31
The FPGA configuration for each application is shown in Table 5. For edge detection, each core had 32 neurons with 128 8-bit synapses per neuron. This allowed 16 cores to be placed on the chip with the main limiter being the number of multipliers on the chip. This limitation is visible in Table 6, where the resource utilization for each application is shown.

For MNIST, 768 layer 1 synapses were needed, thus requiring a different core configuration. 1024 input synapses were utilized per core, with each synapse being 16 bits wide to ensure accurate calculations. A wider bit width was needed because of having a large number of layer 1 inputs. In this design, only 6 cores could be implemented, as the system was limited by the on-chip memory available as shown in Table 6. Table 6 shows the total FPGA resource utilization for these two multicore systems, taking both core and wormhole routing logic into consideration.

<table>
<thead>
<tr>
<th>Table. 5. Network configuration on DE2 board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core utilization</td>
</tr>
<tr>
<td>Neurons per core</td>
</tr>
<tr>
<td>Synapses per neuron</td>
</tr>
<tr>
<td>Data Bitwidth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table. 6. Multicore FPGA resource utilization on DE2 board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
</tr>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>Memory bits</td>
</tr>
<tr>
<td>Multipliers</td>
</tr>
</tbody>
</table>

FPGA performance is compared with an implementation of the applications on a RISC processor. Table 7 shows the throughput achieved on the RISC processor compared to the FPGA. The results show that the FPGA implementation provided about 16× and
300× higher throughput than the RISC processor for the edge detection and MNIST applications respectively.

Table 8 compares the power consumed by the RISC processor and the FPGA. The results show that the RISC processor has an approximately 6× higher power consumption than the FPGA for both application.

Tables 9 and 10 compare the runtime, total energy, and energy delay product of running the two applications for a given set of inputs. For edge detection we examined the processing time for 100 million pixels (these could be divided into a large number of images), while for MNIST character detection, we assumed the processing of 1 million images. The energy delay products show FPGA efficiencies of about 3 and 5 orders of magnitude for the edge and MNIST applications respectively over the RISC core. The edge detection application has a lower efficiency as the RISC core is utilizing convolution applications, while the FPGA is carrying out neural operation (this leads to many more computations on the FPGA). For the MNIST application, both the FPGA and the RISC core are running neural networks, leading to the same amount of computations on both platforms (and hence higher energy efficiencies).

Table. 7. Application throughput on DE2 board

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC</th>
<th>FPGA</th>
<th>FPGA efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.39</td>
<td>6.56</td>
<td>16.70</td>
</tr>
<tr>
<td>(million pixels/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MNIST</td>
<td>1.09</td>
<td>334</td>
<td>306.40</td>
</tr>
<tr>
<td>(thousand digits/sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table. 8. Application power consumption on DE2 board

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC (W)</th>
<th>FPGA (W)</th>
<th>FPGA Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.9</td>
<td>0.145</td>
<td>6.2</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.9</td>
<td>0.147</td>
<td>6.1</td>
</tr>
</tbody>
</table>
### Table 9. Processing 100 million pixels in the edge detector on DE2 board

<table>
<thead>
<tr>
<th></th>
<th>Total time (s)</th>
<th>Total Energy (J)</th>
<th>Energy-Delay (Js)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>254.5</td>
<td>229.0</td>
<td>58,271.7</td>
</tr>
<tr>
<td>FPGA</td>
<td>15.2</td>
<td>2.2</td>
<td>33.7</td>
</tr>
<tr>
<td>FPGA Efficiency</td>
<td>16.7</td>
<td>103.6</td>
<td>1,729</td>
</tr>
</tbody>
</table>

### Table 10. Processing 1 million digits in the MNIST classifier on DE2 board

<table>
<thead>
<tr>
<th></th>
<th>Total time (s)</th>
<th>Total Energy (J)</th>
<th>Energy-Delay (Js)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>917</td>
<td>825.7</td>
<td>757,512</td>
</tr>
<tr>
<td>FPGA</td>
<td>3</td>
<td>0.4</td>
<td>1.3</td>
</tr>
<tr>
<td>FPGA Efficiency</td>
<td>306.4</td>
<td>1,876.1</td>
<td>574,864</td>
</tr>
</tbody>
</table>

#### 6.2.2 Stratix FPGA Results

The multicore neuromorphic processor was also implemented on a GiDEL FPGA board. This contains an Altera Stratix IV FPGA. The system was still programmed in Verilog and compiled by using Quartus II. On this board, a sample application, Edge Detector, is applied to demonstrate the successful operation of the hardware protocol design. Edge detection application means changes or discontinuities in image attributes such as luminance are fundamentally important primitive characteristics of an image because they often provide an indication of the physical extent of objects within the image. An edge detection algorithm is implemented that uses a $3\times3$ mask and generated 8-bit outputs. Input images of any size can be processed. In the neural cores, the algorithm was approximated using a neural network with configuration $9\rightarrow20\rightarrow1$ (9 inputs, 20 neurons in hidden layer, and 1 output neuron).
Table. 11 and Table. 12 shows the configuration of FPGA. Ten cores are utilized in the network. Each core has 64 neurons and 1024 synapses in each neuron. Data bus bit-width is 32 bits. Table 4 demonstrates FPGA resources consumption after compiling the prototyping design in Quartus II. In order to utilize more cores on FPGA, bus bit width is reduced from 32 bits to 16 bits and 16 cores could be utilized shown on Table. 13 and Table. 14. DSP blocks are main limitation from network scaling.

Table. 11. Network configuration with 10 cores on GiDEL board

<table>
<thead>
<tr>
<th>Core utilization</th>
<th>Neurons per core</th>
<th>Synapses per neuron</th>
<th>Bus Bitwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>10</td>
<td>64</td>
<td>32</td>
</tr>
</tbody>
</table>

Table. 12. Multicore FPGA resource utilization with 10 cores on GiDEL board

<table>
<thead>
<tr>
<th>Logic</th>
<th>Registers</th>
<th>Memory bits</th>
<th>18-bit DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>185,920(35%)</td>
<td>82,636</td>
<td>11,081,744 (52%)</td>
</tr>
</tbody>
</table>

Table. 13. Network configuration with 16 cores on GiDEL board

<table>
<thead>
<tr>
<th>Core utilization</th>
<th>Neurons per core</th>
<th>Synapses per neuron</th>
<th>Bus Bitwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
</tbody>
</table>

Table. 14. Multicore FPGA resource utilization with 16 cores on GiDEL board

<table>
<thead>
<tr>
<th>Logic</th>
<th>Registers</th>
<th>Memory bits</th>
<th>18-bit DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>100,928(19%)</td>
<td>79,603</td>
<td>10,151,952 (48%)</td>
</tr>
</tbody>
</table>

Table. 15 shows application runtime and estimated power consumption. The run time of edge detector application is estimated by C++ software on PC. C++ provides
clock() method to calculate run time between program segments. Power consumption is estimated by PowerPlay Early Power Estimator provided by Altera [20].

Table. 15. FPGA run time and power consumption on GiDEL board

<table>
<thead>
<tr>
<th>Run time (ms)</th>
<th>Edge detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (w)</td>
<td>436</td>
</tr>
<tr>
<td></td>
<td>6.24</td>
</tr>
</tbody>
</table>

Fig. 21. Power of total FPGA and detail parts

Table. 16. Application throughput on GiDEL board

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC (million pixels/sec)</th>
<th>FPGA</th>
<th>FPGA efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.39</td>
<td>1.33</td>
<td>3.38</td>
</tr>
</tbody>
</table>

Table. 17. Application power consumption on GiDEL board

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC (W)</th>
<th>FPGA (W)</th>
<th>FPGA Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.9</td>
<td>6.24</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table. 18. Processing 100 million pixels in the edge detector on GiDEL board

<table>
<thead>
<tr>
<th></th>
<th>Total time (s)</th>
<th>Total Energy (J)</th>
<th>Energy-Delay (Js)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>254.5</td>
<td>229.0</td>
<td>58,271.7</td>
</tr>
<tr>
<td>FPGA</td>
<td>75.2</td>
<td>469.2</td>
<td>35,283.8</td>
</tr>
<tr>
<td>FPGA Efficiency</td>
<td>3.4</td>
<td>0.5</td>
<td>1.7</td>
</tr>
</tbody>
</table>
From Table 18, total energy and energy-delay efficiency on Stratix FPGA are lower than RISC simulator [19]. From Table 19, the main result is higher power consumption in I/O part. However, power consumption statistics of RISC simulator does not include any external I/O parts. It is not a fair comparison between Stratix FPGA and RISC simulator. Therefore, I/O power consumption should be subtracted from total FPGA power consumption.

<table>
<thead>
<tr>
<th>Application</th>
<th>RISC (W)</th>
<th>FPGA (W)</th>
<th>FPGA Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge detection</td>
<td>0.90</td>
<td>2.85</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Table. 19. Application power consumption without I/O

<table>
<thead>
<tr>
<th>Total time (s)</th>
<th>Total Energy (J)</th>
<th>Energy-Delay (Js)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>254.5</td>
<td>229.0</td>
</tr>
<tr>
<td>FPGA</td>
<td>75.2</td>
<td>214.3</td>
</tr>
<tr>
<td>Efficiency</td>
<td>3.4</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 20. Processing 100 million pixels in the edge detector without I/O

From Table 20, after subtracting I/O power, better total energy and energy-delay efficiency on Stratix FPGA could be achieved. It shows 3.4x run time efficiency, 1.1x energy efficiency and 3.6x energy-delay efficiency by FPGA implementation.

In order to launch the application on the processor, the first step is sending network configuration and memory initialization data of cores and routers. They are transferred from an external PC to FPGA board by PCI-E interface. On the PC, configuration data was read and generated by C++ and MATLAB software from the origin image. Then network configuration and memory initialization data was transferred to the network module by a simple FIFO. The second step was sending 64,516 groups of input data packets to the network module. After all data packets run through the whole
network, the final output data from the network module was transferred back to the PC by a simple FIFO as well. After receiving the output data, the edge detection processed image was generated by C++ and MATLAB. The output image is shown in Fig. 22.

Fig. 22. Image output from FPGA network
CHAPTER VII
CONCLUSION

This thesis examined the design of a multicore neural processor on both C++ software simulator and hardware prototyping on FPGA. External memory initialization and input transmission system are also established for realizing general purpose. For the two applications examined, the FPGA neural implementation on DE2 provided speedups of 16 and 300 times over the RISC core. In addition, it demonstrated 3.4x run time efficiency and 3.6x energy-delay efficiency by running Edge Detection on Stratix IV FPGA implementation. Very high energy-delay efficiencies were seen for the FPGA. As the future work, network routing will be optimized. State machines in control units will be modified to realize less cycle time cost. Additionally, pipeline data transmission through the network will be explored to make all cores fully utilized in the network. In this way, higher throughput will be achieve by the processor. After FPGA prototyping exploration, an ASIC implementation of these systems would allow significantly higher throughput and lower power consumption.
BIBLIOGRAPHY


