IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORK ON FPGA

Dr. Reza Raeisi[1], Armin Kabir[2]

[1] Indiana State University, Indiana; Email: reza@indstate.edu

[2] Indiana State University, Indiana; Email: akabir1@indstate.edu

1. Introduction

An Artificial Neural Network (ANN) is an information processing paradigm, which is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. Scientists try to imitate the brain's capabilities with ANN (Haykin, 1999). FPGA based reconfigurable computing architectures are well suited to implement ANN because the parallel structure of FPGA matches the topologies of ANN. The ability to easily reconfigure FPGA makes the design less expensive than pre-designed hardware, and it also gives the users the opportunity to change the weights based on their needs (Ienne, 1995). Implemented ANN on FPGA can be used to work with different integrated circuits. For example, ANN is used in cell phones to decrease the noise, in this application ANN should be implemented on the hardware because of the small size of cell phones and also high speed that is needed for this application.

2. Artificial Neural Networks

Some of the most important properties of the human brain that scientists like to imitate are trial-and-error and pattern learning. With ANN, they have come a step closer to reaching this goal.

ANN is used to solve many daily problems in different areas of science. Some of the many applications of ANN are image processing, signal processing, system control, and data mining. The ability of ANN to learn by example makes it very flexible and powerful for numerous applications.

Neurons are the main part of each ANN. Each neuron can have countless inputs; however, it will only have one output value, as illustrated in Figure 2.1.

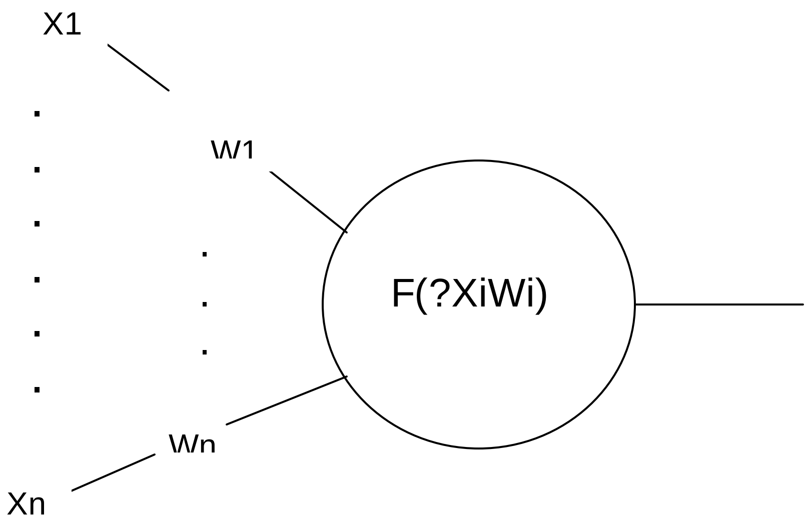

Figure 2.1 Neuron Structure

The output of each neuron is calculated by the following formula:

Output=F(? Xi * Wi) Formula 2.1

Where Xi is the input to the neuron, and Wi is the weight of each input. These weights are calculated after the ANN is trained (Lawrence, 1994).

In Figure 2.2 the structure of an ANN is shown. As it is illustrated, each ANN is made up of neurons and can have three different layers. The input layer, which receives the input from outside the ANN and sends it inward; the output layer, which sends the results of ANN outward; and the hidden layer, which is between the input and the output layers. Most of the processing is done in the hidden layer.

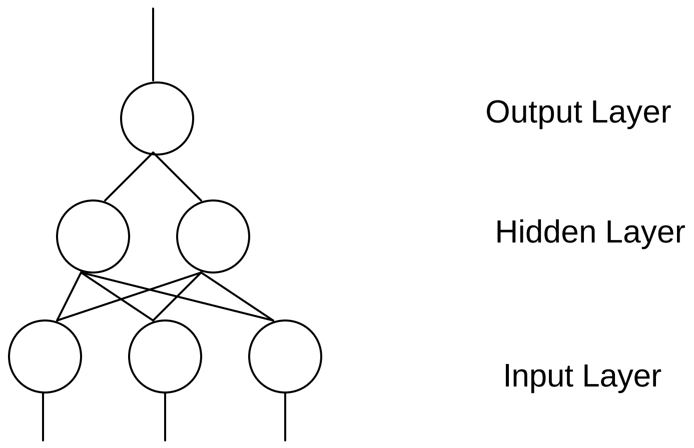

Figure 2.2 ANN Architecture

3. Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) is a specially designed IC that is often used for prototyping. A design engineer can program FPGA on site for a specific application without paying thousands of dollars to have the chip manufactured in mass quantities. With FPGA, designers can reconfigure their design as many times as they want, without the chip manufacturing cost.

Each FPGA has three main parts. The Configurable Logic Block (CLB) is the most important part of each FPGA. CLB provide physical support for the program, which is downloaded on the FPGA. Another part is the Input Output Block (IOB), which provides input and output for FPGA and makes it possible to communicate outside of FPGA. The other part is the Programmable Interconnect, which connects the different parts of FPGA and allows them to communicate with each other. These connections are programmable, so users can define which parts they connect. Figure 3.1 shows the different parts of an FPGA.

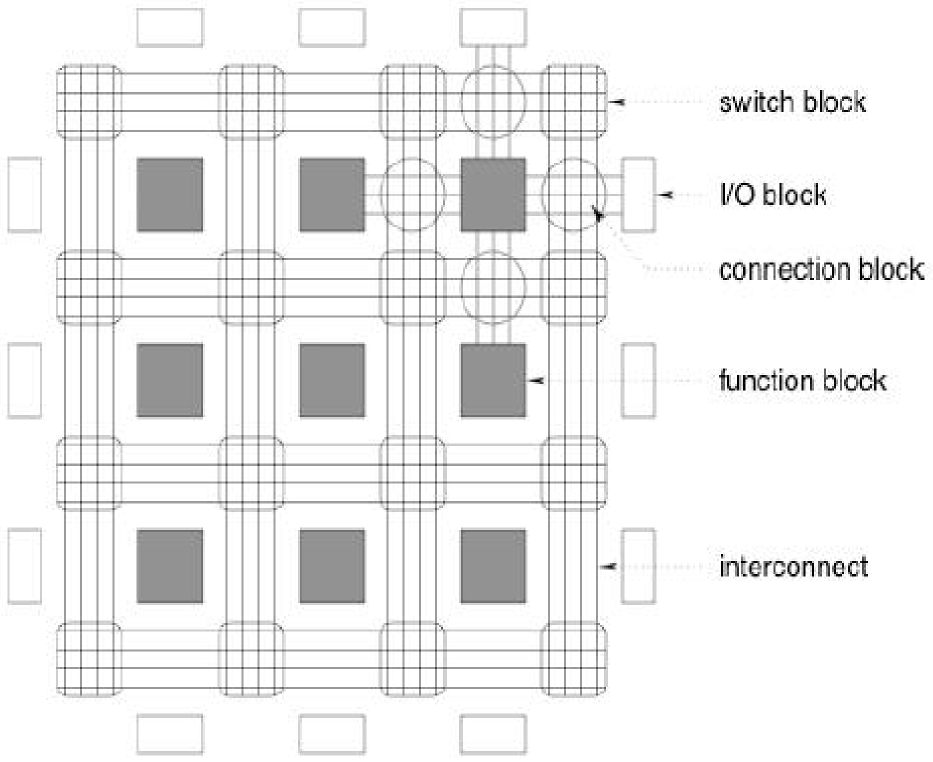

Image Courtesy HTTP://XILINX.COM
Figure 3.1 FPGA structure

4. Implementation

With ANN, weights can be both integers and floating points; however, if an accurate ANN is needed, the weight should be a floating point. Figure 4.1 shows the output of ANN with weights as integers and weights as floating points in comparison with the desired output. As illustrated in this chart, the output of ANN with integer weights is not acceptable because it is inaccurate. However, the output of ANN with floating point weights is accurate and proves that floating weights should be used.

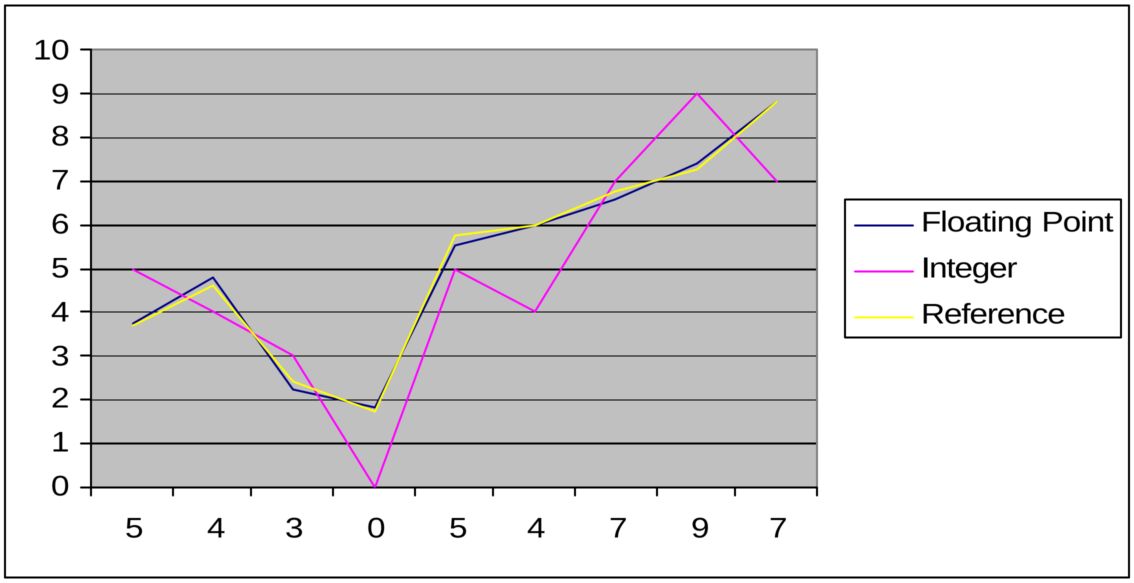

Figure 4.1 ANN with Integer and Floating Point Weights

Different algorithms are available to implement floating point numbers on digital logic. One of the most famous of these is IEEE754 (Shirazi, 1995). In this algorithm each floating point number is mapped into 32 bits: one bit for the sign, 8 bits for the exponent, and 23 bits for the fraction. In this algorithm, the exponent has an offset of 127 in order to show the small number in this standard. The structure of this standard is illustrated in the figure 4.2.

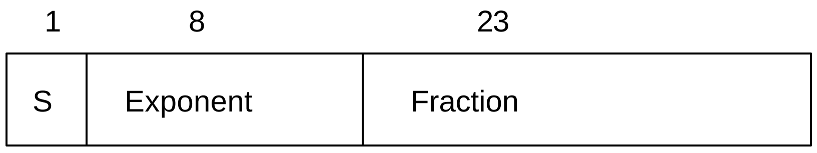

Figure 4.2 IEEE754 Standard

In order to implement ANN, the neuron should be employed first. As shown in formula 2.1, two mathematical functions, addition and multiplication, are needed. The algorithm that is used for the addition of two floating point numbers is illustrated in figure 4.3.

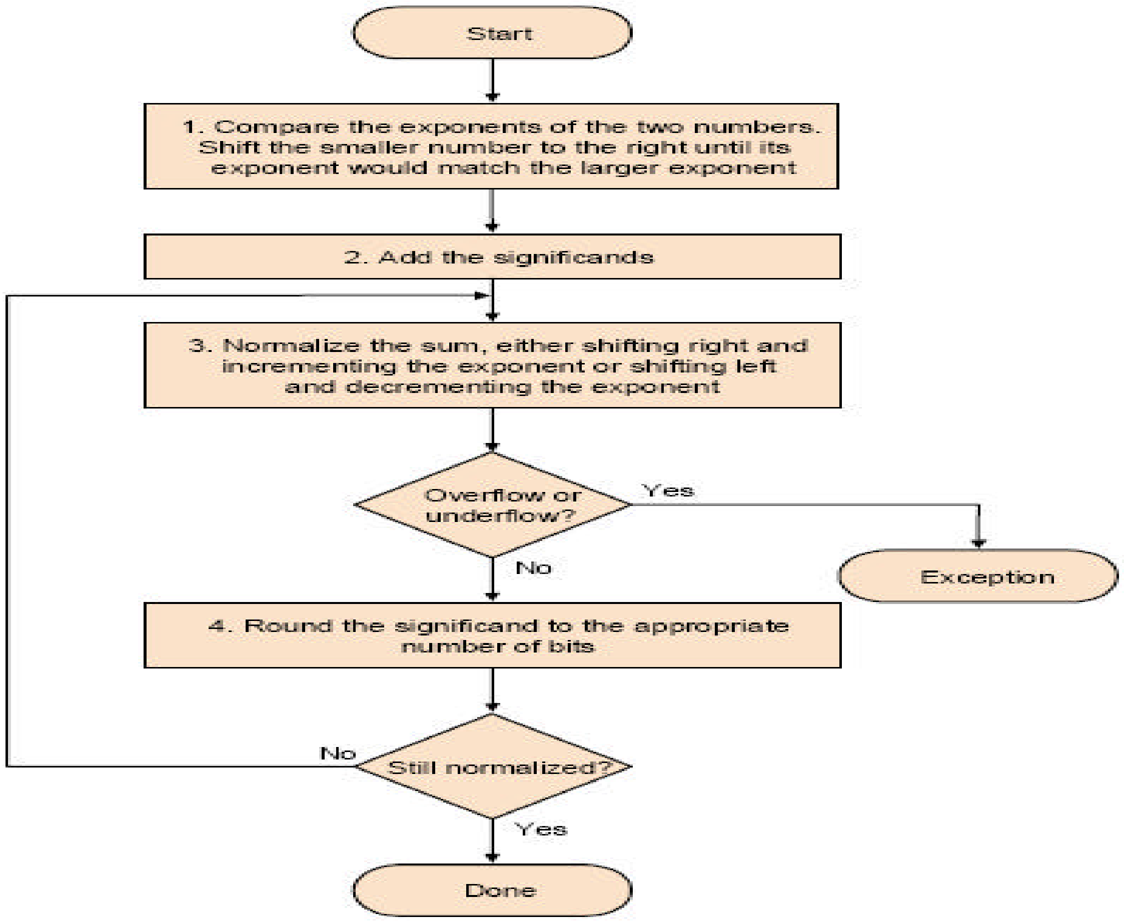
Start
1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significand to the appropriate number of bits
Still normalized?
No
Yes
Done

Image Courtesy Arithmetic for Computer, Louisiana State University (Durresi 2005)
Figure 4.3 Flowchart for Adding to Floating Point Numbers

As it is shown in the flow chart, the exponents of two numbers are compared. The fraction of the number with the smaller exponent is shifted to right until the exponents of the two numbers are equal. After that, the fractions of the two numbers are added. The result will be the exponent of the bigger number. The design that is used for implementing the addition of two floating points in digital circuits is illustrated in figure 4.4.

There are different algorithms available for multiplying two floating point numbers. One is the Parallel Multiplier, which has very high speed and parallel structure; however, it needs a huge number of full adders. For example, to multiply the fractions of two floating point numbers with 23 bits each, the number of full adders needed for this calculation is 529, which is beyond the current FPGA capacity (Smith, 2001).

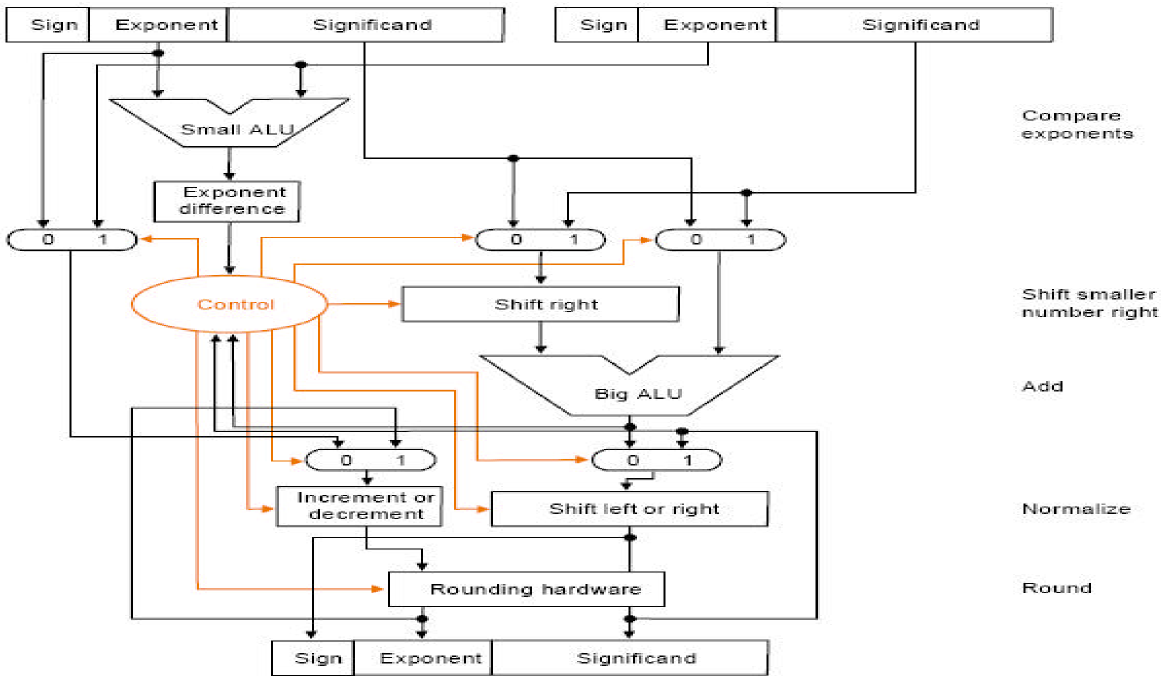

Image Courtesy Arithmetic for Computer, Louisiana State University (Durresi 2005)
Figure 4.4 Digital algorithm of adding two floating point numbers

Another algorithm that can be used for multiplying two floating point numbers is the Serial Parallel Multiplier. This algorithm is slower than the Parallel Multiplier, but the number of full adders that it needs to implement is much less. The Serial Parallel Multiplier only uses 23 full adders, which is drastically lower than the 529 that the Parallel Multiplier needs. The algorithm of the Serial Parallel Multiplier is illustrated in figure 4.5. As shown in this figure, the multiplier will be sent to the circuit serially, and the multiplicand will be sent to circuit in parallel. The result of the multiplication will be in serial mode (Belanovic, 2002). As shown, the number of full adders that are needed for multiplication is considerably reduced by this algorithm.

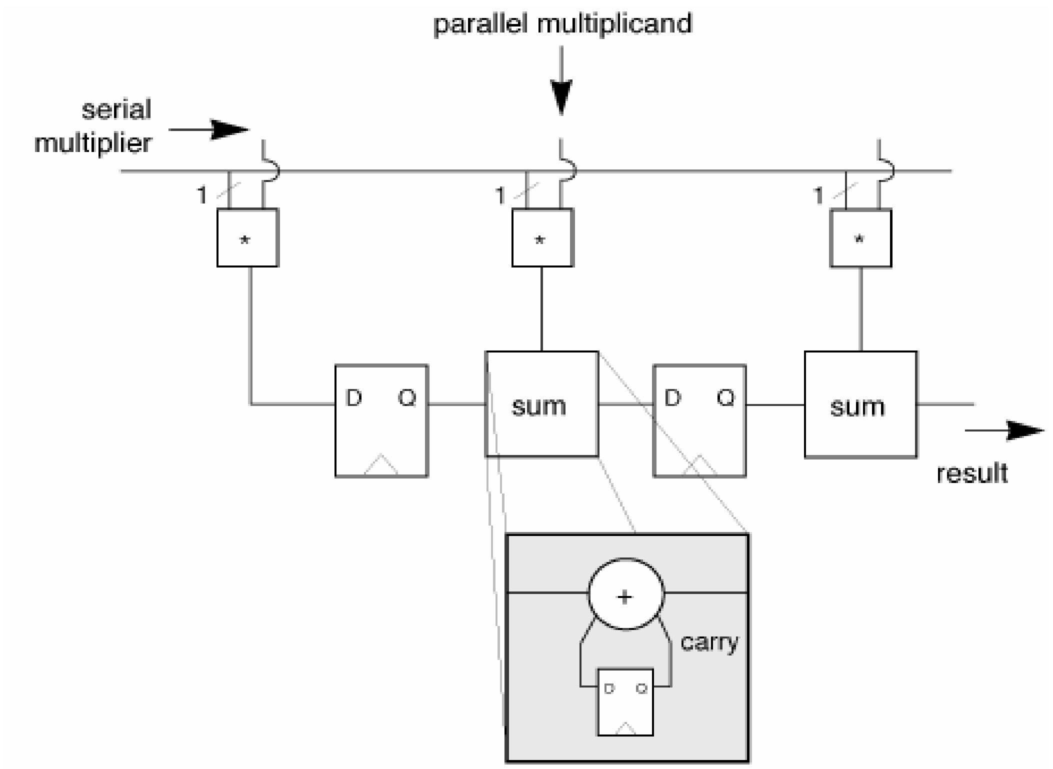

Image Courtesy VLSI Design: System on Chip Design (Wolf 2004)
Figure 4.5 Serial Parallel Multiplier

To increase the performance of the design, it is possible to change the number of bits in the fraction of IEEE754. In order to increase the performance of the design, the required accuracy for ANN should be calculated first. Next, the number of bits in the fraction which are needed to provide the planned accuracy should be calculated. Figure 4.6 shows the comparison between the speed and the number of bits in the fraction for the add function. As shown in the chart, by increasing the number of bits in the fraction, the speed of the system will decrease, but the accuracy will improve.

Therefore, by completing the hardware of both the adder and the multiplier, the Artificial Neural Network construction on FPGA was implemented.

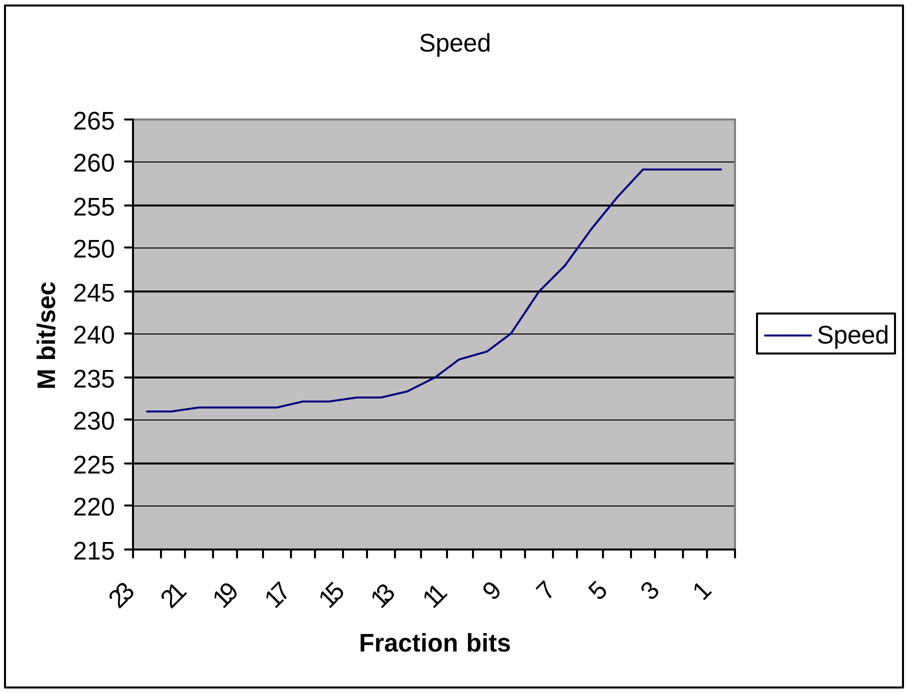

Figure 4.6 Accuracy versus Speed

5. Result

This project is supported by an internal grant through “The Alliance for Excellence through Engagement and Experiential Learning” at Indiana State University. The purpose was to experiment and demonstrate to students the development and implementation of Artificial Neural Networks (ANN) with Field Programmable Gate Array (FPGA) based on Floating Point Multipliers and Adders.

We provided a specific hardware chip solution for real time applications incorporating ALtera and Xilinx FPGA boards. A total of nine neurons were implemented on the Spartan II FPGA with a speed of 13 M bits/sec. In order to implement bigger ANN on FPGA it is possible to use several FPGA on each layer, as shown in figure 5.1. This research project provided the specific method and understanding of the hardware of neural networks and its relation with Floating Point Multipliers and Adders for better precision.

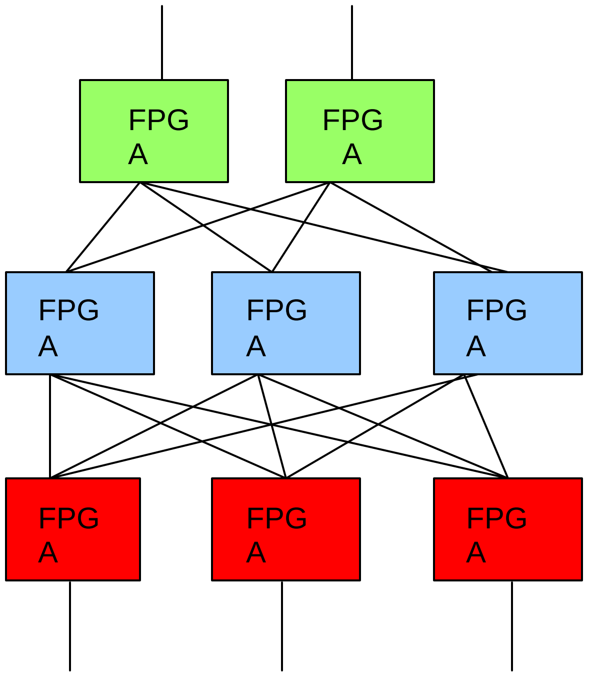

Figure 5.1 ANN with more than 9 neurons

References

Belanovic P. (June 2002) *Library of Parameterized Hardware Modules for Floating-Point Arithmetic with An Example Application*, M.S. Thesis, Dept. of Electrical and Computer Engineering, Northeastern University.

Dr. Durresi A. (2005) Arithmetic for Computer, Louisiana State University. Web Site: http://bit.csc.lsu.edu/~durresi/CSC3501_05/3_3501_05_2.pdf

Haykin S. (1999). *Neural Networks, A Comprehensive Foundation* (2th Ed). New York: Prentice Hall.

Ienne P. (June 1995) *Digital hardware architectures for neural networks, SPEEDUP Journal*, vol. 1, n. 9.

Lawrence J. (1994). *Introduction to Neural Networks: Design, Theory, and Applications* (6th Ed). Los Angeles: California Scientific Software.

Mencer O., Morf M., and Flynn J. (1998) *High Performance FPGA Design for Adaptive Computing.* IEEE Symposium on FPGAs for Custom Computing Machines, Napa, CA.

Shirazi N., Walters A., and Athanas P. (Apr. 1995) *Quantitative Analysis of Floating Point Arithmetic on FPGA-based Custom Computing Machines*. In IEEE Workshop on FPGAs for Custom Computing Machines, pages 155–162, Napa, CA.

Smith D. , Franzon (2001). *Verilog Style for Synthesis of Digital System*. New York: Prentice Hall.

Wolf W. Modern (Jan 2004) *VLSI Design: System on Chip Design(3rd)*. Prentice Hall.

Xilinx (2002) ISE Logic Design Tools. In *http://www. xilinx. com/xlnx/xil prodcat landingpage.jsp?title= Design+Tools*.