

# FPGA-Based Stochastic Neural Networks –Implementation

Stephen L. Bade and Brad L. Hutchings  
Dept. of Electrical and Computer Eng.  
Brigham Young University  
Provo, UT 84602  
Email: hutch@ee.byu.edu  
Tel: (801) 378-2667

April 12, 1994\*

## Abstract

*Reconfigurable Field-Programmable Gate Arrays (FPGAs) provide an effective programmable resource for implementing hardware-based Artificial Neural Networks (ANNs). They are low cost, readily available and reconfigurable –all important advantages for ANN applications. However, FPGAs lack the circuit density necessary to implement large parallel ANNs with many thousands of synapses. This paper presents an architecture that makes it feasible to implement large ANNs with FPGAs. The architecture combines stochastic computation techniques with a novel lookup-table-based architecture that fully exploits the lookup-table structure of many FPGAs. This lookup-table-based architecture is extremely efficient: it is capable of supporting up to two synapses per Configurable Logic Block (CLB). In addition, the architecture is simple to implement, self-contained (weights are stored directly in the synapse), and scales easily across multiple chips.*

## 1 Introduction

Reconfigurable Field-Programmable Gate Arrays (FPGAs) provide a reprogrammable hardware resource that can be exploited to great advantage for Artificial Neural Networks (ANNs). In contrast with custom Very Large Scale Integration (VLSI), they provide many important advantages. They are readily available at reasonable cost and have a reduced hardware development cycle. In addition, FPGA-based ANNs can be tailored to specific ANN configurations; there

is no need for worst-case fully-interconnected designs as in full-custom VLSI design.

The fundamental problem limiting the size of FPGA-based ANNs is the high cost of implementing multiplication. High-speed, i.e., fully-parallel, ANNs require a tremendous number of multipliers –each synaptic connection in an ANN requires a single multiplier and the number of synaptic connections typically grows as the *square* of the number of neurons. For example, a fully-interconnected, 10-neuron ANN requires 100 multipliers (10 synapses per neuron x 10 neurons); the same network with 100 neurons would require 10,000 multipliers.

Practical implementations of ANNs are possible only if the amount of circuitry devoted to multiplication is significantly reduced. This can be accomplished in two ways: 1) reduce the number of multipliers, and/or 2) reduce the complexity of the multiplier. Most digital, hardware-based ANNs reduce *both* multiplier complexity and the total number of multipliers. For example, digital ANNs typically allocate a single bit-serial multiplier per neuron and use time-division multiplexing (TDM) to share it across all neuron inputs [3]. Performance is reduced but this multiplexed approach is much more practical for moderately sized ANNs.

Another way to reduce the circuitry necessary for multiplication is to use bit-serial *stochastic* computing techniques. Stochastic computing uses relatively long, probabilistic, bit-streams where the numeric value is proportional to the density of “1”s in the bit-stream. The main advantage of stochastic computing is that the multiplication of two probabilistic bit-streams can be accomplished with a single 2-input logic gate. This makes it feasible to implement large, dense, fully-parallel networks with digital techniques that achieve a high-level of performance.

---

\*Presented at IEEE Workshop on FPGAs for Custom Computing Machines Workshop, Napa, CA, April 10-13, 1994, pg. 189-198

Our paper presents new and original work in the following areas of application and theory:

- *Architecture.* This paper presents a lookup-table-based architecture that fully exploits the table-lookup organization of many FPGAs. This architecture can achieve densities (neurons per FPGA component) that are at least an order of magnitude higher than the architecture presented in [7]. Moreover, it uses the FPGA resources such that it is optimal in the number of logic-block outputs used. This makes it feasible to implement very large ANNs with FPGAs. For example, it is possible to implement 480 synaptic connections on a single Xilinx 4010 part with this architecture.
- *Activation function.* A simple and effective activation function can be computed by thresholding a stochastic sum.
- *ANN Implementation.* A single-chip ANN is implemented using which can process 181K patterns per second.

## 2 Stochastic ANNs: A Brief Review

In general, all ANN architectures consist of a set of inputs and interconnected neurons, some of which are output neurons. This is shown in Figure 1. Each

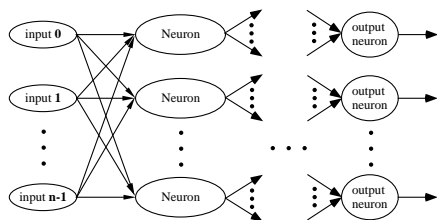


Figure 1: Overall Network Architecture

neuron contains a set of synapses and an activation function (see Figure 2). The function of the synapse is to multiply its input by a weight value to produce an output. A neuron’s activation function is a non-linear thresholding function on the sum of the output of its synapses.

A stochastic ANN uses the same general organization described above; the primary difference is that it uses stochastic techniques for signal representation and computation. Signals are represented as relatively long bit-streams where the actual signal value is encoded as a statistical distribution of the “1”s in the

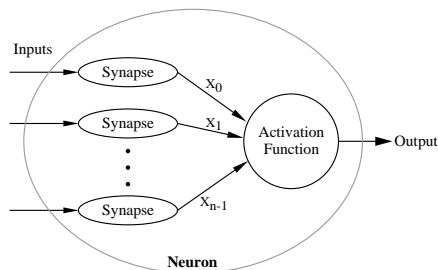


Figure 2: Neuron Structure

bit-stream. Computation is then based upon principles of statistics and uses simple boolean functions to perform the otherwise complex operations of multiplication, thresholding, and activation. The main advantage of the stochastic technique is that it can dramatically reduce the circuitry required to implement a synapse and neuron, and in turn, an entire neural network.

The stochastic ANN will be presented in two parallel sections: a conceptual view and an implementation view. The conceptual view presents the general theory and organization behind the stochastic ANN architecture. The implementation view discusses the specific details of ANN implementation with lookup-table FPGAs, specifically, the Xilinx FPGA. Finally, simulation results of an entire network are presented.

## 3 The Stochastic ANN : Conceptual View

This section presents the inputs first, followed by the synapse and the activation function.

### 3.1 Inputs

The first step toward stochastic processing is to convert real-valued inputs into a stochastic bit-streams<sup>1</sup>. The converted bit-streams are produced by a *bit-stream generator*. For a description of bit-stream generators, see [6, 7]. For the purposes of this paper, a bit-stream generators is circuit elements that uses a Linear Feedback Shift Register (LFSR).

### 3.2 The Synapse

The function of the synapse is to multiply some input by an internally stored weight value. This mul-

<sup>1</sup>This conversion need only occur at the interface between the world and the ANN; within the ANN itself, *all* signals are represented as stochastic bit-streams.

tiplication is performed using stochastic bit-streams. A stochastic bit-stream is simply a sequence of equally weighted bits where the probability of each bit being set is proportional to the value the bit-stream represents. More formally, let  $A$  be a Bernoulli random variable, and  $A(t)$  be the value of  $A$  at time  $t$ . Then the sequence of values  $[A(0), A(1), \dots, A(n-1)]$  is a stochastic bit-stream. In this paper, capital letters are used to represent both Bernoulli random variables and the bit-streams they generate.

There are two common mappings of a real number  $e$  to a Bernoulli random variable. These mappings directly affect the synapse circuitry, and so are presented here.

*Representation I. Unipolar:* The unipolar representation maps rational number  $e$  in the interval  $(0, v)$  to the random variable  $A$  with generating probability:

$$p = P(A = 1) = e/v \quad (1)$$

Thus the maximum value ( $e = v$ ) would be when every bit of the bit-stream is 1, and the minimum ( $e = 0$ ) when each bit is 0. Stochastic multiplication with this representation is a simple bitwise “and” of the input bit-streams. For example, given the real values  $x$ ,  $y$ , and  $z$ , ( $0 \leq x, y \leq v$ ), where  $xy = z$ , the value  $z/v$  can be obtained by the “and” of the bit-streams that represent  $x$  and  $y$ . Let the generating probabilities for the bit-streams that represent these values be  $p_x, p_y$ , and  $p_z$  respectively. The “and” operation defines

$$p_z = p_x p_y \quad (2)$$

Since  $p_i = e_i/v$ , this gives

$$z/v = xy \quad (3)$$

Note the product of the multiplication is scaled by  $1/v$ . This scaling keeps the range the same for of numbers in the ANN.

*Representation III. Bipolar:* A more compact bipolar representation is to map a rational number  $e$  ( $-v \leq e \leq v$ ) to the binary random variable  $B$  with generating probability:

$$p = p(B = 1) = e/2v + \frac{1}{2} \quad (4)$$

Thus the maximum value ( $e = v$ ) is reached when every bit of the bit-stream is 1, and the minimum ( $e = -v$ ) when each bit is 0, and zero when exactly half the bits are 1. The inverse transform

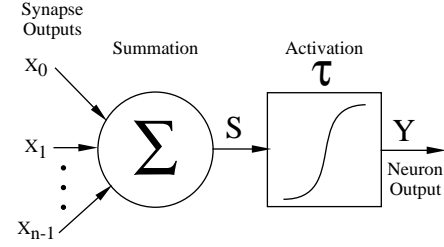


Figure 3: Activation Function

is  $e/v = 2p - 1$ . Multiplication of bipolar bit-streams is the bitwise “x-nor” of the input bit-streams. In this representation, given the real values  $x$ ,  $y$ , and  $z$ , ( $-v \leq x, y \leq v$ ), where  $xy = z$ , the value  $z/2v$  can be obtained by the “x-nor” of the bit-streams that represent  $x$  and  $y$ . Again let the generating probabilities for the bit-streams that represent these values be  $p_x, p_y$ , and  $p_z$ . The “x-nor” operation defines

$$p_z = p_x p_y + (1 - p_x)(1 - p_y) \quad (5)$$

Substituting  $p_i = e_i/2v + \frac{1}{2}$ ,

$$z/v = xy \quad (6)$$

Thus synaptic multiplication of two bit-streams can be performed with a single boolean logic function. The following chapter on implementation presents an approach in which the synapse stores a compact representation of the weight, and then *generates the actual weight bit-stream during computation*.

### 3.3 Activation Function

The second stage of neural processing consists of summation and activation. During summation, the sum  $s(t)$  is computed from the weighted inputs from the synapses. The activation function,  $\mathcal{T}$ , maps  $s(t)$  to  $Y$ , the neuronal output. For this architecture, the activation function, is a continuous sigmoid-like function that is achieved by simply outputting a “1” if the sum of the synapses is above a threshold value, and a “0” if it is not.

Assume that at clock cycle  $t$ , a Bernoulli random variable (or bit-stream)  $B$  takes the value  $B(t) \in \{0, 1\}$ . Let the synapse outputs be variables  $X_0, X_1, \dots, X_{m-1}$ , and the neuron output be  $Y$ , then the neuronal output is written as:

$$Y(t) = \mathcal{T}(s(t)) = \mathcal{T}\left(\sum_{i=0}^{m-1} X_i(t)\right) \quad (7)$$

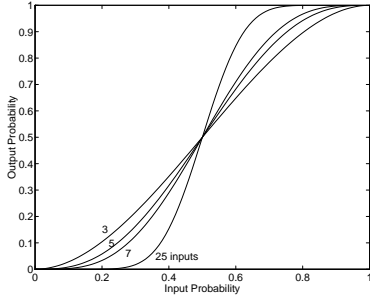


Figure 4: Binomial Activation Function

where

$$\mathcal{T}(s(t)) = \begin{cases} 1 & \text{if } s(t) > \frac{m}{2} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

The simple threshold function above depends on the stochastic nature of  $s(t)$  to generate a sigmoid-like output. Notice that the value of  $s(t)$  is an integer between 0 and  $m$  (number of neuron inputs), so  $s(t)$  cannot be represented by a Bernoulli random variable. We have chosen to represent the random vector  $[X_0, X_1, \dots, X_{m-1}]$  as  $\mathbf{S}$ , which has a value of  $s$  (number of ones in  $\mathbf{S}$ ). At each clock cycle  $t$ ,  $\mathbf{S}(t)$  is the  $m$  length vector  $[X_0(t), X_1(t), \dots, X_{m-1}(t)]$ .

To describe the activation function, it is necessary to explain the distribution of  $\mathbf{S}$ . Assume for the moment that all the bit-streams  $X_i$  have the same generating probability ( $p$ ). Then the vector  $\mathbf{S}$  will have a binomial distribution described by  $p$  and  $m$  (length of  $\mathbf{S}$ ), so the probability that  $\mathbf{S}$  will have  $k$  bits set is:

$$P(s(t) = k) = \binom{m}{k} p^k (1-p)^{m-k} \quad (9)$$

The probability value of the output  $Y$  is:

$$P(Y(t) = 1) = \mathcal{T}(s(t)) = \sum_{k=\lceil \frac{m}{2} \rceil}^m \binom{m}{k} p^k (1-p)^{m-k} \quad (10)$$

Figure 4 gives a graphical view of this function for a 3, 5, 7, and a 25 input neuron. The function becomes more non-linear as the number of inputs increases. The threshold function is continuous, and is very easy to realize in hardware. It is continuous because the independent variable ( $p$ ) is continuous, even though the actual values represented by a finite length bit-stream are quantized. The quantization depends on the length of the output bit-stream  $Y(t)$ .

Now drop the assumption that each input bit-stream to the activation function has the same probability. Then  $\mathbf{S}(t)$  no longer has a binomial distri-

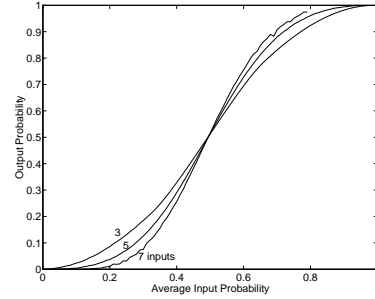


Figure 5: Actual Threshold function

bution. Thus Equation 10 no longer accurately describes the threshold function. This is obvious by noting that Equation 10 is a function of one probability only, whereas each random variable in  $\mathbf{S}$  has a *separate* probability. Thus the activation function is not a function of one input, i.e., the average probability of all  $m$  inputs, but is a function of  $m$  *distinct* inputs.

The actual function is:

$$P(Y(t) = 1) = \mathcal{T}(s(t)) = \sum_{j=0}^{2^m-1} \mathcal{T}(j) \prod_{i=0}^{m-1} P(X_i = \Phi_{ji}) \quad (11)$$

where

$$\Phi_{ji} = \begin{cases} 1 & \text{if bit } i \text{ in } j \text{ is set}^2 \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

and

$$\mathcal{T}(j) = \begin{cases} 1 & \text{if } > \frac{m}{2} \text{ bits in } j \text{ are set} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

Simulations have shown the actual function to be in general slightly more non-linear than the approximation, which should work well for a neural network. This function is shown in Figure 5.

## 4 The Stochastic ANN : Implementation

As stated in the introduction, the ANN implementation works well with custom VLSI, but is optimized for dynamically reconfigurable FPGAs. The architecture was developed specifically for Xilinx FPGAs, but should work well with any lookup-table-based FPGA. A brief overview of the Xilinx FPGA architecture is given so that implementation details can be discussed. Following this the implementation of each part of the ANN will be presented.

<sup>2</sup>treating  $j$  as a binary number.

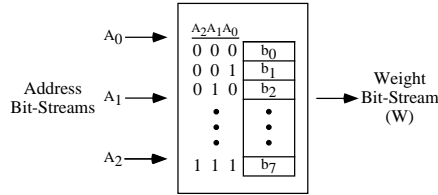


Figure 6: Lookup Table to Generate Synapse Weight

#### 4.1 Xilinx FPGA Organization

The basic building block of a Xilinx FPGA is the Configurable Logic Block, or CLB. A CLB has two D flip-flops, can implement two logic functions of four variables or one function of five variables, and has two outputs. These logic functions are implemented as 32-bit lookup tables. Because the logic functions are implemented with table lookup, there is no penalty for complicated logic functions as opposed to simple ones. Circuits are implemented by programming and interconnecting the CLBs. CLB utilization is typically limited by the number of flip-flops available, by the number of logic functions, or by the number of outputs available. Currently, Xilinx FPGAs have from 64 to 480 CLBs per chip [4]. Functionality per CLB will be our metric for circuit density.

#### 4.2 Synapse

As stated in section 3.2, the synapse not only does a multiply, but it also stores a weight value and generates a weight bit-stream. Because synapse multiplication is a simple logic function of 2 inputs, the size of the circuitry that generates the weight bit-stream determines the size of the synapse.

The architecture described here proposes a novel approach to weight bit-stream generation and stochastic multiplication. This approach minimizes the synapse circuitry and maps very efficiently to lookup-table-based FPGA architecture. Rather than treating bit-stream generation and multiplication as discrete operations, this approach implements the entire synapse as a *single boolean function of four variables*. This approach provides several distinct advantages over other implementations. First, the required circuitry is drastically reduced because it exploits the lookup-table architecture common to FPGAs, and second, no control circuitry is required which further reduces circuit requirements. These advantages make it possible to construct practical implementations of large self-contained ANNs with FPGAs.

#### 4.2.1 Generating the Weight Bit-Stream

Consider the problem of generating a synapse weight bit-stream. This can be performed as a boolean function of three variables. This function is implemented with an eight by one lookup table (eight locations of one bit each) that encodes both the weight value and the logic for generating the weight bit-stream (see figure 6). Three *address bit-streams* serve as inputs to the lookup table, represented by the Bernoulli random variables  $A_0$ ,  $A_1$ , and  $A_2$ .

These address bit-streams determine the precision with which the weight can be represented. With  $n$  lookup table bits,  $\log_2 n$  to  $n$  bits of precision can be achieved. The weight bit-stream ( $W$ ) is made of the bits ( $b_7, b_6, \dots, b_0$ ) in the table that are addressed by the address bit-streams, as shown in Figure 6. Let the triplet  $A_2A_1A_0$  represent a three digit binary number, then the probability that bit  $i$  will be addressed is  $P(A_2A_1A_0 = i)$ . The weight bit-stream can be described by:

$$P(W = 1) = \sum_{i=0}^7 b_i P(A_2A_1A_0 = i), \quad b_i \in \{0, 1\} \quad (14)$$

Thus the available weight values are dependent on the probability values of the lookup table bits. For example, if each bit had an equal probability of being addressed (of  $\frac{1}{8}$ ), then only nine distinct weights ( $0, \frac{1}{8}, \frac{2}{8}, \dots, \frac{7}{8}, 1$ ) could be generated. This would correspond to 3 bits of precision. The optimal bit-probability of being addressed is  $\frac{2^n}{2^m - 1}$  where  $n$  is the bit position, and there are  $m$  total bit positions. With this bit probability, the lookup table could be thought of as an unsigned binary number with an implied decimal point before its most significant bit. This would allow 256 weight bit-streams to be generated ( $0, \frac{1}{255}, \frac{2}{255}, \dots, \frac{254}{255}, 1$ ). Choosing values for the synapse address bit-streams is discussed in [1, sect 4.3].

The address bit-streams are easily generated by the same method used for the input bit-streams. This is done by hard-coding the bit-stream generator to produce only one value. In addition, multiple uncorrelated sets of synapse address bit-streams may be derived by using delayed versions of a single set of address bit-streams. This is possible because Bernoulli values  $A(i)$  and  $A(j)$  are uncorrelated for  $i \neq j$  [5].

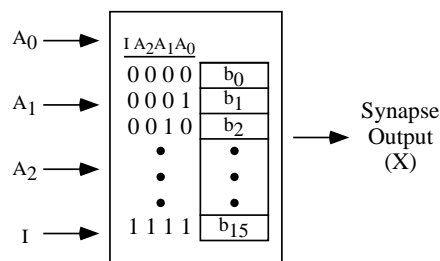


Figure 7: Complete Synapse

#### 4.2.2 Multiplying the Input with the Weight Bit-Stream

The synapse output is the product of the weight bit-stream,  $W$ , and the synapse input,  $I$ . This multiplication is a logic function of two variables ( $W$  and  $I$ ), and the weight is a function of three variables ( $A_0$ ,  $A_1$ , and  $A_2$ ). These can be combined as one function of four variables ( $A_0$ ,  $A_1$ ,  $A_2$ , and  $I$ ), as shown in Figure 7. The next step is to determine the lookup-table value for the synapse. Let  $I$  be the most significant address bit for the lookup table, and the contents of the 8-bit lookup table that generates the desired weight be the vector  $\mathbf{B} = [b_7, \dots, b_0]$ . The synapse lookup table is then the 16-bit vector  $[\mathbf{B}|\mathbf{0}]$  for the unipolar representation, and  $[\mathbf{B}|\mathbf{B}]$  for the bipolar representation. This corresponds to an “and” of  $W$  and  $I$  for the unipolar representation, and a “x-nor” for the bipolar representation.

### 4.3 Activation Function

In this architecture, the summation and activation functions for an  $m$ -input neuron are implemented directly as a boolean function of  $m$  inputs. This is possible due to the simple way in which the activation function thresholds the synapse outputs. Like the synapse, this approach has important advantages: 1) it maps very well to lookup-table based FPGAs, 2) the activation function operates at the same speed as the synapse, so that neither becomes a bottleneck for the other, and 3) no control circuitry is required. Unlike the synapse implementation, the size of the activation function hardware varies with the number of inputs to the neuron.

For small numbers of inputs, the activation function can be implemented directly in a lookup table with  $m$  inputs,  $2^m$  lookup table bits, and one output. For five or fewer inputs, this can be implemented with a single CLB. For large numbers of inputs, this direct implementation quickly becomes impractical be-

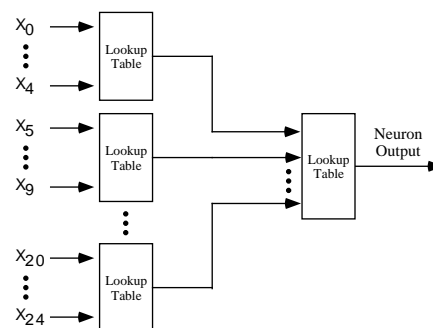


Figure 8: 25-Input Activation Function

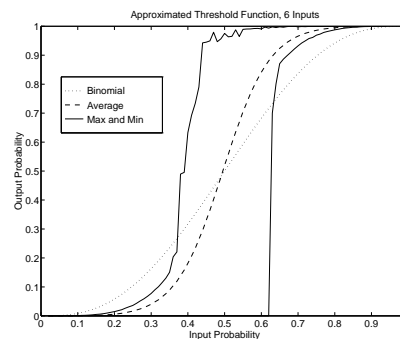


Figure 9: Approximated Activation Function

cause of the size of the required lookup table. For large numbers of inputs, we approximate the activation function by partitioning the function of  $m$  inputs ( $m > 5$ ) into a set of smaller thresholding functions of 5 or less inputs<sup>3</sup>. The outputs of these smaller functions are then thresholded to produce the neuronal output. Figure 8 depicts how this would be done for a 25 input neuron. With up to five inputs per CLB, the size of the activation function using this approximation tends towards  $1/4m$  CLBs for an  $m$ -input neuron.

This approximation does introduce some error into the activation function. Figure 9 shows the average and bounding paths that the activation function can take for an 6-input neuron. The error is minimized when the inputs to the activation function assume a binomial distribution.

<sup>3</sup>Functions are limited to five variables or less because a single Xilinx CLB can directly implement functions of up to five inputs.

## 5 The Complete Network

A complete ANN using this architecture consists of the following:

- One Linear Feedback Shift Register to provide the random carrier bit-streams for both the inputs and the synapse address bit streams.
- One bit-stream generator for each input to the system.
- Three bit-stream generators to provide the address-bit-streams used by all the synapses in weight generation.
- Delay elements (flip-flops) used to provide delayed versions of the inputs and address bit-streams to reduce correlation.
- Neurons, which consist of synapses and activation functions, and
- Circuitry to convert the output bit-streams to usable values. This can be provided with a simple counter or integrator, depending on the application.

Figure 10 gives a complete view of the network. Notice that each synapse has four inputs, one from the previous layer, and three address bit-streams. A network can be configured with any number of layers; the figure depicts a two-layer network for the sake of brevity.

The variable that most affects the size of the network is the amount of correlation allowed in the network. Correlation introduces error in the neural processing, but eliminating it entirely greatly increases the network's size. Correlation can be eliminated completely by providing each synapse a unique delayed version of the address bit-streams, and a unique delayed version of it's input. These four delays add a cost of two CLBs (four flip-flops) to each synapse, making the total synapse size two and one-half CLBs rather than only one-half CLB. Reasonable accuracy may be obtained while heavily reusing bit-streams. We found that using a total of  $m$  synapse address bit-streams works well (one for each input), as shown in Figure 10.

### 5.1 Accuracy

Given a Bernoulli sequence of length  $n$  from a random variable with generating probability  $p$ , it is shown

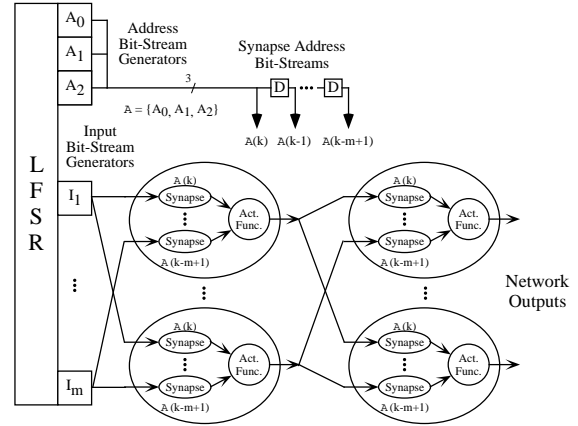


Figure 10: Complete Network

by [2, pg. 58] that the standard deviation of the estimated value of  $p$  is

$$\sigma(\hat{p}) = \left[ \frac{p(1-p)}{n} \right]^{\frac{1}{2}} \quad (15)$$

Thus the average error of a sequence is at a maximum for  $p = 0.5$  and it decreases with the square root of the length of the sequence sampled. Thus for the worst case, to achieved  $b$  bits of accuracy, you need a sequence of length  $2^{2b-2}$ . [7]

An important observation is that all of the operations performed in the network produce Bernoulli sequences if they have Bernoulli sequences as inputs. The only errors in excess of that discussed above in the network are due to the correlation of input bit-streams.

## 6 Implementation

We had three implementation goals: First, a working software model which simulated the architecture *bit-for-bit* as it would run in hardware, second, to show that the dense ANNs could be built using this architecture, and third, to build and test a working “proof of concept” circuit. The hardware implementations will be presented first, followed by some results from the software simulator. To demonstrate the density that could be achieved, an ANN was completely routed. The individual CLBs were not programmed, as the goal was simply to show that routing was possible.

This was implemented on a smaller Xilinx part from the 4000 series, the 4003PG120-5, which has 100 CLBs laid out in a 10x10 array. The 4000 series was chosen over the 3000 series because of the far superior routing

resources of 4000 series. An attempt was made at using the 3000 series (using the 3090pc84), but the limitations in routing resources did not allow as dense of a layout as had been hoped.

## 6.1 Circuit Description

The circuit implemented a single layer ANN. The layer is fully connected with 12 inputs and 10 outputs. More layers could be added by simply cascading this circuit. This circuit consists of 120 synaptic connections (120 stored weights, with weight bit-stream generation and 120 multipliers), and 10 activation functions. The inputs to the chip are the neuron inputs (as bit-streams) and the synapse address bit-streams. The inputs could be generated either on or off-chip. As our intent is to show the neural processing circuitry, we will assume the input bit-streams are generated off-chip. The outputs are the ten bit-streams from the activation functions. Because the activation function has more than 5 inputs, the function is computed in two steps, as discussed in section 4.3 (see Figure 11). Three sub-activations of four inputs each are computed, and then they are run through an activation function to produce the neuronal output.

In the physical layout, each neuron with it's 12 synaptic connections occupies one row on the chip, or 10 CLBs. This is depicted in Figure 11. Aside from the synapse weights each row is identical, all receiving the same inputs and synapse address bit-streams. Each row consists of three identical blocks each containing four synapses and sub-activation circuitry. These blocks occupy three CLBs each, two for the synapses and one for the sub-activation function. This uses nine of the 10 CLBs per row. The 10th CLB is used to compute the overall activation function for the row. Figure 11 shows a row of the physical layout.

As shown by Figure 11, each of the synapse address bit-streams are shared by two synapses. Sharing address lines was done simply to reduce pin and routing requirements. This does cause some correlation, which was computed in software simulations of the circuit. [1, Sect. 6]. The correlation of the inputs to the activation function was measured both with separate and shared synapse address bit-streams. The correlation of the bit-streams entering an activation function was found by taking the difference of the observed joint probability and the product of the marginals. An overall sum squared error was then computed using each of the joint probabilities. For example, if an activation function had two inputs,  $A$  and  $B$ , then the sum

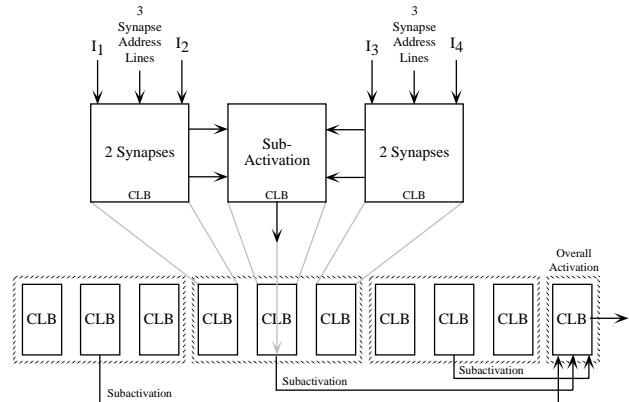


Figure 11: One row of the Physical Layout

squared error would be:

$$\sum_{i=0,1} \sum_{j=0,1} [P(A = i, B = j) - P(A = i)P(B = j)]^2 \quad (16)$$

The sum squared error was typically between .01 and .06 as computed by the software simulator.

As long as the synapse weights are not too similar, the correlation does not noticeably increase when the synapse address lines are shared. Nor did sharing address lines noticeably impact the output of the activation function. The correlation may cause accuracy problems if the inputs and synapse weights are too similar, and there may be correlation problems with multi-layer networks. This is an area of future research.

This design allows 120 synapses on only 100 CLBs. Every CLB on the chip is fully utilized. 60 CLBs are used to implement the synapses, and 40 CLBs are dedicated to activation function circuitry. This approaches the upper limit of synapse-density that is possible on this style of FPGA.

In order to make the best use of the resources on the chip, it was necessary to have more control over the physical routing than is normally available with high level design tools. The design was routed using XDE (an interactive physical layout design tool by Xilinx). XDE allowed hand routing which was necessary to place and route the circuit. One of the most useful routing resources for this design was the long lines that run the width and length Xilinx FPGAs. The long lines enabled the input and synapse address lines to run the length of the chip to all the synapses. The problem with routing this design on the 3000 series is that there are only a few long lines with limited connections, which made routing impossible. A hierarchy of macros were used to route the entire chip.

One difficulty we observed in routing this design is that the design tools currently available do not provide good support for allowing the designer to specify the physical layout. Such a tool would be very helpful if one was to try and do very much work designing layouts such as this.

## 6.2 Timing and Circuit Speed

Since the output of each CLB is gated through a flip-flop, the determining factor for the speed of the device is the routing delays. The main factor for the delay through a net is the number of "switchboxes" the net goes through. The nets that went through the most switchboxes had the greatest delays, with the longest delay being 27.6 ns (timing calculated using `xdelay`). Nets with the same fan-out (20 outputs) that used long-lines rather than going through switchboxes had delays of up to 6ns. A more careful route could avoid the longer delays by using long-lines rather than switch-boxes, and operate at much higher speeds. With delays of up to 27.6ns, the part can operate at 36Mhz.

The speed measurement of Connections Per Second (cps) is not a good measure of parallel machines such as this, since the number is so dependent on the size of the network. For example, if assuming a bit-stream length of 200 bits (3.5% error), one input vector (pattern) could be processed in  $5.5\mu\text{s}$ , or about 22 Mcps. A larger network using the same architecture (say 12000 synapses) would run at 2.2 Gcps. Thus a better metric for a parallel ANN is the number of patterns it can process per second (PPS), regardless of pattern size or network size. Using this metric, our design can operate at 181 KPPS.

## 6.3 Proof of Concept Circuit

The second goal was to produce a small working proof of concept circuit. This circuit implemented a three input, one output, single layer network. The circuit was implemented in a Xilinx 3090-pc84 FPGA, and installed in the National Technology Inc. Gips-II board, which allows the FPGA to be configured and tested with a PC. The longest routing delay for this circuit was 16ns. The circuit was tested with random input patterns and generated the expected outputs.

## 6.4 Simulation Results

The software simulator was mainly a test-bed for developing this architecture. It was designed to collect error and correlation statistics. This section presents

findings concerning neuron output error and the correlation of the weighted inputs to neurons. The bit-stream error is calculated by taking the absolute value of the difference of the expected probability and the observed bit-stream probability, for example, given a bit-stream  $A$  of length  $n$  with expected value of  $p$ , the error is calculated by:

$$error = \left| p - n^{-1} \sum_{i=0}^{n-1} A(i) \right| \quad (17)$$

Thus an error of .08 would correspond to the bit-stream value being 8% different than expected. The error is caused by both the variance of the binomial distribution (Section 5.1) and by correlation. The correlation reported in the following graphs is found by taking the sum squared of the differences of the joint probability and the product of the marginals for each joint probability.

For these simulations, a fully-connected three-layer, feed-forward network with five inputs was simulated. The network configuration had five neurons per layer with five inputs per neuron, for a total of 15 neurons and 75 synapses. Simulations were run with bit-stream lengths of 100, 250, 500, 1000, and 2500. 100 trials were run with each of these bit-stream lengths, 50 with a unipolar representation, and 50 bipolar. Each trial had a new random set of synapse weights. Simulation time (for all 500 trials) was under an hour on an i486-50MHz PC.

Figure 12 shows the maximum and average error values for all 15 neurons in the network. The two dashed lines are the maximum errors for each bit-stream length, one line for the unipolar representation, and one for the bipolar representation. The knees in the graph are due to sparse data points on the x axis. The average neuron error is represented by two dotted lines on top of each other, one for each number representation. This shows that the number representation does not significantly affect the neuron output error. The solid line is the predicted maximum error assuming no correlation, as given by equation 15, with  $p = 0.5$ .

As explained earlier, correlation is introduced only to reduce the hardware required to implement the ANN. Thus the tradeoff is network size v.s. loss of accuracy due to correlation. For a specific application, the network could be designed with as much or as little correlation in order to achieve the required accuracy. Defining this tradeoff more formally could be addressed in further research.

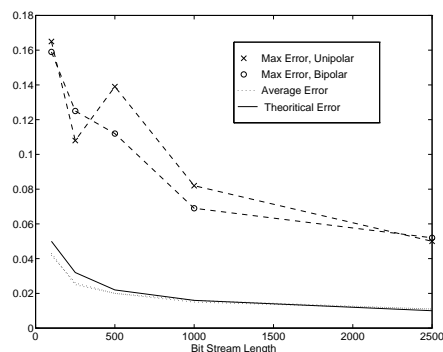


Figure 12: Maximum and Average Error of Neuron Output

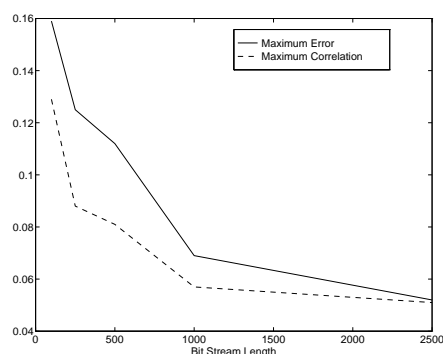


Figure 13: Relationship between Error and Correlation

## 7 Conclusions

This paper has presented a lookup-table-based architecture for stochastic ANNs that maps efficiently to FPGAs. It exploits all of the advantages inherent to FPGAs: low cost, availability, reconfigurability, the ability to support arbitrary network architectures, lowered development costs, all while providing a level of density that makes it feasible to implement large ANNs with FPGAs. This architecture provides the following distinct advantages:

- The entire network is self-contained, there is no need for supporting circuitry to store weights or aid in computations.
- The structures are very regular and repeated.
- There is no control logic except the clock.
- There are no bus lines and delays for reading weight values, since each synapse stores it's own weight.

- The architecture is fully parallel, resulting in  $O(1)$  operation.
- The architecture scales easily, and can be scaled across multiple chips to form large networks.

This architecture also provides the greatest possible synapse density on the above described Xilinx FPGA architecture. Circuits implemented with Xilinx FPGAs, for example, are often output limited because each CLB provides only two outputs, one per function generator. This architecture uses one function generator and one output per synapse, making it possible to implement two synapses with a single CLB. Even if the function generator could be eliminated, one output per synapse would still be required, therefore two synapses would still consume an entire CLB.

## 8 Future Work

The main issue which needs to be addressed is network learning algorithms. Learning is complicated in stochastic networks because the number representation is such that it is hard to store a number with enough precision to perform standard learning calculations. We would also like to study the ability of the architecture to support very large networks, for example, networks with 10,000 - 100,000 synapses or more. Networks of this size could be constructed with Multi-Chip Modules (MCMs) or Wafer-Scale Integration (WSI). The reprogrammability of FPGAs is strong advantage for WSI implementations because it makes it possible, in many cases, to program *around* defects in the wafer. Another important area of study is the effects that correlation has on accuracy. Allowing some correlation provides a denser network at the cost of some accuracy. It will be important to understand the tradeoffs between correlation and accuracy so that the accuracy of a given network can be predicted.

## References

- [1] Stephen L. Bade. Lookup table based neural network using FPGAs. *Thesis, Brigham Young University*, 1994.
- [2] B. R. Gaines. Stochastic Computing Systems. *Advances in Information Science*, 2(1):37–172, 1969.
- [3] Dan Hammerstrom. VLSI Architecture for High Performance, Low Cost, On-Chip Learning. *IJCNN-9-Wash-DC*, 2:537 – 544, January 1990.
- [4] Xilinx Inc. *The Programmable Logic Data Book*. 1993.
- [5] Robert J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.
- [6] M. van Daalen, P. Jeavons, J. Shawe-Taylor, and D. Cohen. Device for Generating Binary Sequences for Stochastic Computing. *Electronic Letters*, 29(1):80 – 81, 1993.
- [7] Max van Daalen, Peter Jevons, and John Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, April 1993.