N94-18355

VLSI Synthesis of Digital Application Specific Neural Networks

Grant Beagles
Department of Electrical Engineering
Montana State University
Bozeman, Montana 59717

Kel Winters
Advanced Hardware Architectures, Inc.
Moscow, Idaho 83843

Abstract- **Neural networks tend to fall into two general categories, 1) software simulations, or 2) custom hardware that must be trained. The scope of this project is the merger of these two classifications into a system whereby a software model of a network is trained to perform a specific task and the results used to synthesize a standard cell realization of the network using automated tools.**

1 Introduction

Neural net research may be roughly classified into two general categories; software simulations or programmable neural hardware [2,6].

Many neural network simulators are readily available. The major drawback to all of them is that, no matter how well written, they are run on a sequential machine. This means that the software must simulate the parallelism of the network and slows down dramatically as the number of connections increases [1,3].

Hardware neural networks are usually general purpose and must be trained. Depending on the training, a significant percentage of the total hardware resources may be unused. By defining the network with a software model and then synthesizing the network from that model, all of the silicon area will be utilized. This should result in a significant reduction in die size when comparing the application specific version to a general purpose neural network capable of being trained to perform the same task.

2 Network Modeling

The simulator that is being used for this project is version 2.01 of **NETS** written by Paul T. Baffes of the Software Technology Branch of the Lyndon B. Johnson Space Center [3]. This simulator was chosen for several reasons. **NETS** has a flexible network description format, the source code is available, and the weight matrix may be stored in an ASCII file for easy use in later steps.

As a first design effort, a simple numeral recognition network with three layers and 37 neurons was defined. The network consists of a 5 by 6 input layer, one hidden layer that is also 5 by 6, and a 1 by 7 output layer. This network is fully connected. Figure 1 contains the **NETS** description of the network.

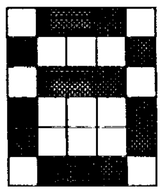
```
LAYER : 0                --INPUT LAYER
        NODES : 30
                X-DIMENSION : 5
                Y-DIMENSION : 6
        TARGET : 2

LAYER : 1                --OUTPUT LAYER
        NODES : 7
                X-DIMENSION : 1
                Y-DIMENSION : 7

LAYER : 2                --FIRST HIDDEN LAYER
        NODES : 30
                X-DIMENSION : 5
                Y-DIMENSION : 6
        TARGET : 1
```

NETS description of neural network.

Figure 1.

A training set consisting of ten digits (0 through 9) and the corresponding ASCII values is used to build the network weighting matrix. Figure 2 illustrates a typical character representation and its corresponding input/output vector. The network training set does not include any noisy or corrupted data to simplify the model. Training the network required 100 iterations and was completed in about 6 minutes. The fully connected network has 1110 connections. The weights in the weight matrix range between ±1.7 following training.

Once the network is trained, the number of connections is reduced. This is done by setting all weights having an absolute value less than a specified value to zero (no connect). This process is easily automated allowing various cut off values to be evaluated. The modified weight matrix is evaluated using **NETS** to determine whether or not the network will still satisfactorily perform its designed task. Table 1 is a summarizes the results of reducing the network.



011 1000 b

(.1 .9 .9 .9 .1
.9 .1 .1 .1 .9
.1 .9 .9 .9 .1
.9 .1 .1 .1 .9
.9 .1 .1 .1 .9
.1 .9 .9 .9 .1
.1 .9 .9 .9 .1 .1 .1)

Character w/ASCII code — Input/output vector

Example of training set element.
Figure 2.

CUT-OFF VALUE	NUMBER OF CONNECTIONS	SATISFACTORY PERFORMANCE	THRESHOLD[1]
0.3	688	yes	0.5
0.4	530	yes	0.5
0.5	413	yes	0.5
0.55	344	yes	0.5
0.6	288	no	----

[1] Any value ≥ threshold is a "one" otherwise "zero".

Table 1.

The actual cut-off values tested ranged up to 1, however, all results with a cut-off above 0.55 were inconsistent with the desired results. Figure 3 is the test vector for the character shown in figure 2 with its associated output vector. (The cut-off is 0.55 and the threshold is 0.5.)

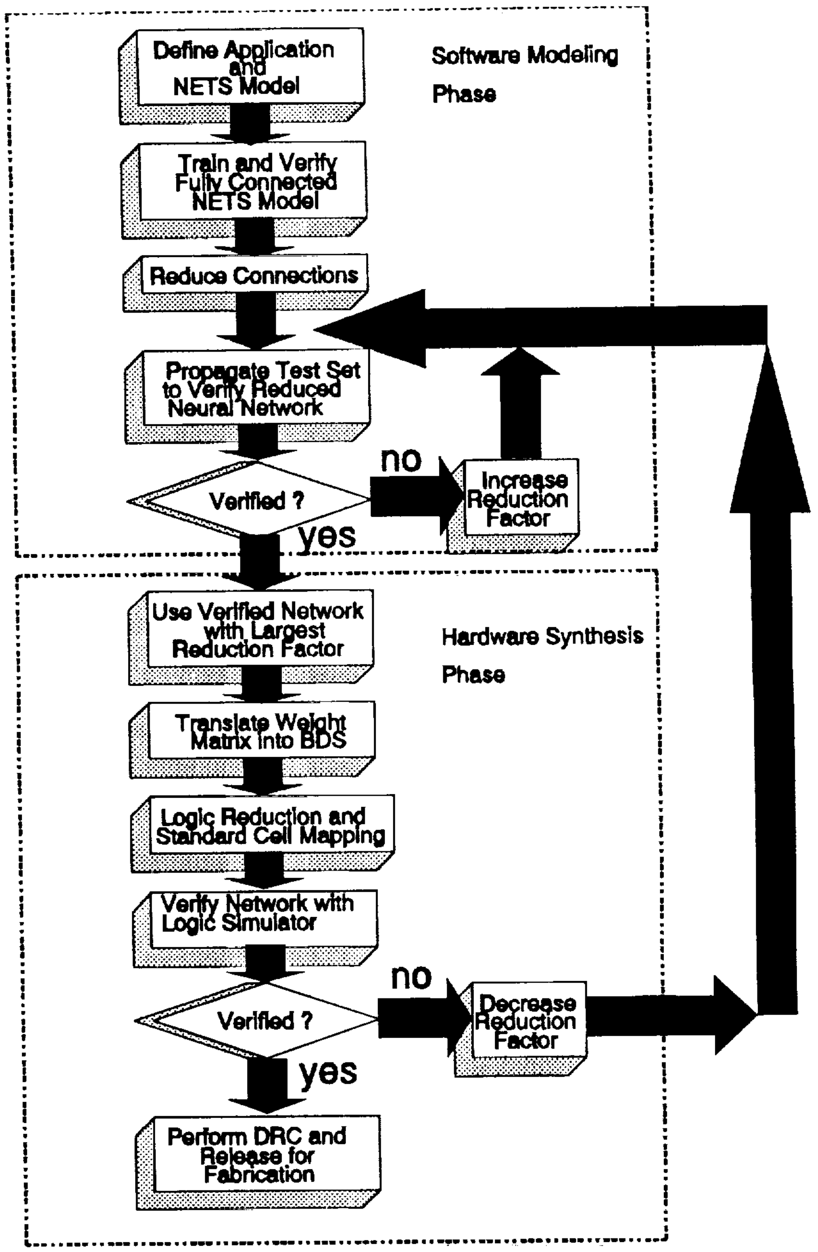
```
-- test set for min.net

(.1 .9 .9 .9 .1-- "8"
 .9 .1 .1 .1 .9
 .1 .9 .9 .9 .1
 .9 .1 .1 .1 .9
 .9 .1 .1 .1 .9
 .1 .9 .9 .9 .1)

Outputs for Input 8:

( 0.002   0.846   0.994   0.865   0.036   0.257   0.164)
```

Output vector for test vector from figure 2.
Figure 3.

With the threshold value taken into consideration the output is 011 1000, which is the ASCII code for "8".

3 Logic Synthesis

The intent of the neural network synthesis process is to provide a fully automatic path to silicon realization once a network model has been constructed and verified in the **NETS** environment. The entire synthesis process is schematically shown in figure 4. The **OCT** tool set from the University of California, Berkeley [8], was chosen for the back end of this procedure, which includes logic optimization, technology mapping, standard-cell place-and-route,and composite artwork assembly and verification.


Define Application and NETS Model
Software Modeling Phase
Train and Verify Fully Connected NETS Model
Reduce Connections
Propagate Test Set to Verify Reduced Neural Network
Verified ?
no
Increase Reduction Factor
yes
Use Verified Network with Largest Reduction Factor
Hardware Synthesis Phase
Translate Weight Matrix into BDS
Logic Reduction and Standard Cell Mapping
Verify Network with Logic Simulator
Verified ?
no
Decrease Reduction Factor
yes
Perform DRC and Release for Fabrication

Application specific neural network synthesis process.
Figure 4.

First, the completed neural network topology is translated from the **NETS** environment to the **OCT** hardware description language **BDS** by a *NETS-to-OCT* program written for this purpose. A simple example of a single neuron in **NETS** netlist and the corresponding BDS description are shown in figure 5. The **BDS** file is then compiled into unminimized logic functions by the **OCT** tool **Bdsyn**. These are mapped into a standard cell library by **MisII**. Currently, the **SCMOS2.2** standard-cell library from Mississippi State University is used, implemented in the **SCMOS6 N-Well CMOS** process available from the National Science Foundation **MOSIS** program. This process has a minimum feature size of two microns.

```
LAYER : 0--INPUT LAYER
NODES : 5
TARGET : 1


LAYER : 1--OUTPUT LAYER
NODES : 1
```

Majority logic NETS description.

```
MODEL dumb
        out<0>,sum0<4:0>=in<4:0>;

ROUTINE dumbnet;

!       target layer # 0  node # 0

sum0<4:0> =     8
                + in0<0>
                + in0<1>
                + in0<2>
                + in0<3>
                - in0<4>

IF sum0<4> EQL 1
        THEN out<0> = 1
        ELSE out<0> = 0;
ENDROUTINE;
ENDMODEL;
```

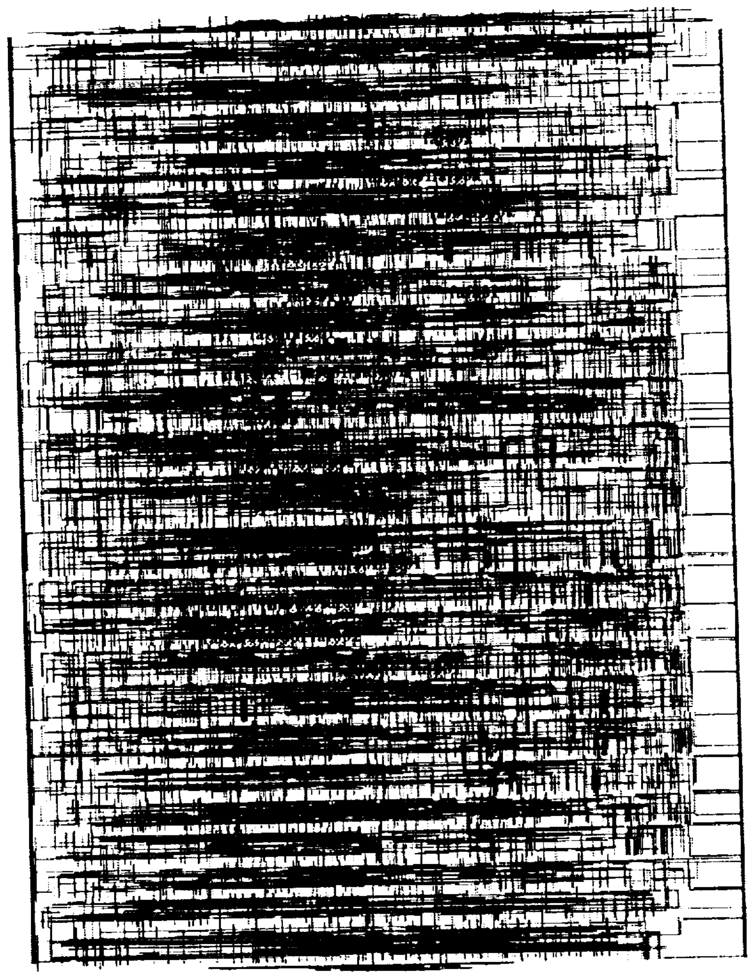
Majority logic BDS description.

Figure 5.

MisII is an *n-level* logic optimizer, which creates a realization of a logic function from a given cell library minimizing both worst-case propagation delay and the number of cells required. The relative priority of area versus speed is user selectable. The result is stored in the OCT database and may be verified with **MUSA**, a multilevel simulator in the **OCT** suite. From here, the design process may be easily iterated from the **NETS** description forward as shown in figure 4.

A number of additional **OCT** tools are available for padring composition, composite placement and channel routing, power distribution routing, and artwork verification. Artwork may be generated from the **OCT** database in Caltech Intermediate Format (**CIF**) for release to **MOSIS** or other foundry services.

The standard-cell realization of the digit recognizer described previously is shown in figure 6. Its 37 neurons required 2741 standard cells in 47 square millimeters.

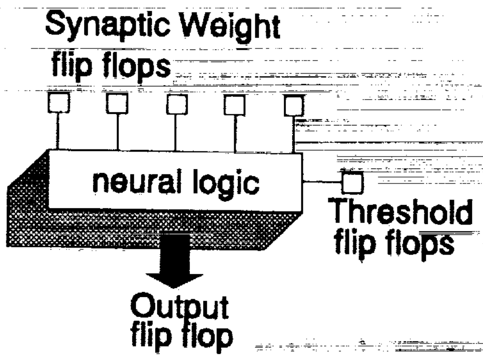


Standard cell realization of character recognizer.

Figure 6.

4 Conclusions and Future Directions

Figure 7 shows a block diagram of a 5 input programmable neuron. To build the digit recognizer using this generic neuron would require about 45 neurons. The actual network has 37 neurons. The increased number of generic neurons is due to the five input limitation. Many of the nodes in the network have more than five inputs. With the generic neurons, multiple neural cells would be connected at the outputs giving the behavior characteristics of a neuron having a larger number of inputs. The number of standard cells required for the entire network realized with the generic 5 input neuron is approximately 8280 (~45 neurons by 184 standard cells per neuron [7]). This network would cover nearly 141 square millimeters.



Generic five input neural cell.
Figure 7.

As stated previously, the network created with the methodology described here requires 2741 standard cells and 47 square millimeters. This represents a 66% reduction in the number of cells used and silicon area. This reduction will allow the chip to be fabricated at a significantly lower cost than a chip with a sufficient number of the generic neurons. Furthermore, all of the silicon area in the application specific area is utilized whereas, a significant percentage is unused in the general model. These results are very preliminary. Experiments with simpler models suggest that substantial improvements in standard cell optimization remain possible.

The models used in this research were trained using ideal training sets, meaning that the characters were well formed and the level of contrast between the background and the characters was high. For the neural network to have any real value, a larger training set would be necessary. This set would have both poorly formed and low contrast examples of each character. Using a training set of this type would cause an increase in the number of connections necessary in the network [5].

The synthesis process described may be used to deliver an application specific neural network, trained to perform a specific task at less cost than utilizing general neural hardware. Silicon area will be more highly utilized in the application specific case since only the necessary circuitry is fabricated. Although more research is necessary, early results show the method to be promising.

Acknowledgement

This work was supported by an educational grant from the National Science Foundation MOSIS program and equipment donations from the Hewlett-Packard Co, Tektronix Inc., and Advanced Hardware Architectures Inc. The authors would especially like to thank Paul Cohen, of Advanced Hardware Architectures; Andrea Casotto of UC Berkeley; Dr. Gary Harkin, Jaye Mathisen, Diane Mathews, and Bob Wall, of Montana State Univerisity; and Dr. Gary Maki, of the University of Idaho; for their invaluable assistance.

References

[1] H. C. Anderson, "Neural Network Machines," *IEEE Potentials*, Vol. 8 no. 1, pp. 13-16, Feb 1989.

[2] J. A. Anderson,D. Hammerstrom, and L. D. Jackel, "Neural Network Applications for the 90's," *IEEE Videoconference*, May 23, 1991.

[3] P. T. Baffles, *NETS User's Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX.

[4] M. W. Firebaugh, *Artificial Intelligence*, Ch. 18, PWS-KENT, Boston MA, 1988.

[5] H. P. Graf, L. D. Jackel, W. E. Hubbard, "VLSI Implementation of a Neural Network Model," *IEEE Computer*, pp. 41-49, March 1988.

[6] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, Vol 1 and 2, MIT Press, Cambridge, MA, 1986.

[7] S. Wandler and D. Metcalf, "Development of a Neural Network Integrated Circuit," Senior Project Report, Montana State University, Department of Electrical Engineering, 1991.

[8] A. Casotto, ed., *OCTTOOLS Revision 5.1 User Guide* , University of California Berkeley, Electronics Research Laboratory, Berkeley, CA, 1991.