Artificial Neural Network

DSD Project Final Report

James Letendre and Ziyan Zhou

5/12/2010

Artificial Neural Networks(ANN) are non-linear statistical data modeling tools, often used to model complex relationships between inputs and outputs or to find patterns in data. In this project, a generic ANN is designed and implemented to run on the Altera DE2 FPGA board.

Abstract

Artificial Neural Networks (ANN) are non-linear statistical data modeling tools, often used to model complex relationships between inputs and outputs or to find patterns in data. In this project, a generic hardware based ANN is designed and implemented in VHDL. This three-layer ANN is implemented entirely with 32-bit single precision floating point arithmetic to guarantee flexibility and accuracy for its wide range of applications. The ANN supports reconfigurable numbers of perceptron per layer as well as supervised learning through back propagation. Mean squared error is used to measure the quality of learning as part of the Built-in Self-Test (BIST). An rudimentary example application is implemented to demonstrate the capability of the ANN on an Altera DE2 FPGA board. The example application is a fully functional pattern recognizer, built by utilizing the ANN. This classifier is trained to identify letters on a 4x4 binary grid populated by a user through 16 toggle switches. The most probable class suggested by the ANN is displayed on a LCD screen.

Algorithm

An artificial neural network is consists of perceptrons, usually organized into three layers: input layer, hidden layer, and output layer. Perceptrons (j) in a given layer are individually connected to each of the perceptrons (i) from the previous layer, with a weight of $w_{i,j}$ placed on the connection. Since each perceptron is a nonlinear function acted on the weighted sum of outputs from other perceptrons, by changing the weight on each connection in an ANN, any smooth function can be approximated.

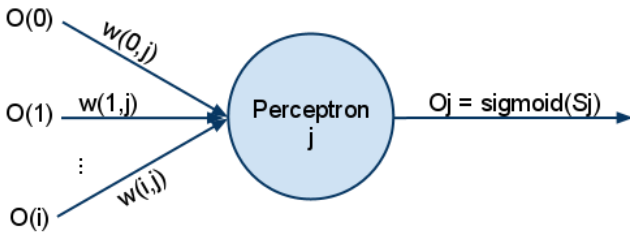

Figure 1: Structure of a perceptron

Feed Forward

The feed forward process of a perceptron (j) works as follow:

1. A weighted sum is calculated from all input connections of the perceptron.

$$s_j = \sum_{i=0}^{n} w_{i,j} o_i$$

2. The weighted sum is then passed through an activation function, usually sigmoid.

$$o_j = \frac{1}{1 + e^{-s_j}}$$

3. The output of this activation function is then fed forward into the next layer where it will be used to calculate the weighted sum for each of the perceptrons. The perceptron output for the output layer is then regarded as the neural network's output.

Back Propagate

The back propagation learning process for a perceptron (j) works as follow:

1. Find the error of output for the perceptron. If the perceptron is located in the output layer, this error is simply the difference between the output and the learning target (t_j). Otherwise, the error is computed from a weighted sum of forward layer's errors (e_o) and the derivative of the sigmoid function.

$$e_j = \begin{cases} t_j - o_j, \text{ if m is in the output layer} \\ o_j\,(o_j + 1)\sum_{o=0}^{l} w_{j,o}\, e_o\,, \text{ otherwise} \end{cases}$$

2. The change of weight for each input connections for the perceptron is then computed from the error of the perceptron and the previous value of the connection. A learning rate λ is applied such that overall error can be slowly minimized. λ often has the value of 0.05.

$$\Delta w_{i,j} = -\lambda\, o_j e_j, \text{ where } \lambda \text{ is the learning rate}$$

3. New weight is then saved for each of the input connections of the perceptron.

Design

The ANN is designed to be generic such that the number of perceptrons per layer is easily reconfigurable to meet the requirement of a specific application.

Since individual weights are assigned to each connection, a size of (N_I + 1) * N_H + (N_H + 1) * N_O array is needed for the storage of all connection weights, where N_I is defined as the number of perceptrons in the input layer, N_H is defined as the number of perceptrons in the hidden layer, and N_O is defined as the number of perceptrons in the output layer. As the number of perceptrons in a network grows, the number of weight storage needed becomes exceedingly large. Due to the limited resource on the FPGA itself, this ANN is designed to offload such burden of memory space to an external SRAM where storage space is relatively cheaper.

Note that in this ANN design, each perceptron also receives a bias connection. The bias connection is like any other connection except that its value is not an output of a perceptron from a previous layer but set to the constant 1. The weight for this bias connection is also updated in the back propagation process. And it helps make the neural network more effective.

The ANN is also designed to accept an external floating point ALU to allow maximum flexibility. It is recommended to use a hardware built-in floating point ALU if one is present. Otherwise, one can be synthesized from VHDL code.

A 16-bit linear feedback shift register is also required for the ANN to function properly. This LFSR provides necessary randomized initial values for all the connections between perceptrons. Experiments showed that a randomized initial network greatly reduce the training length required to reach the same degree of learning. The ANN automatically generates weights between -1.0 to 1.0 using the 16-bit random numbers.

Below is a block diagram of the ANN design component:

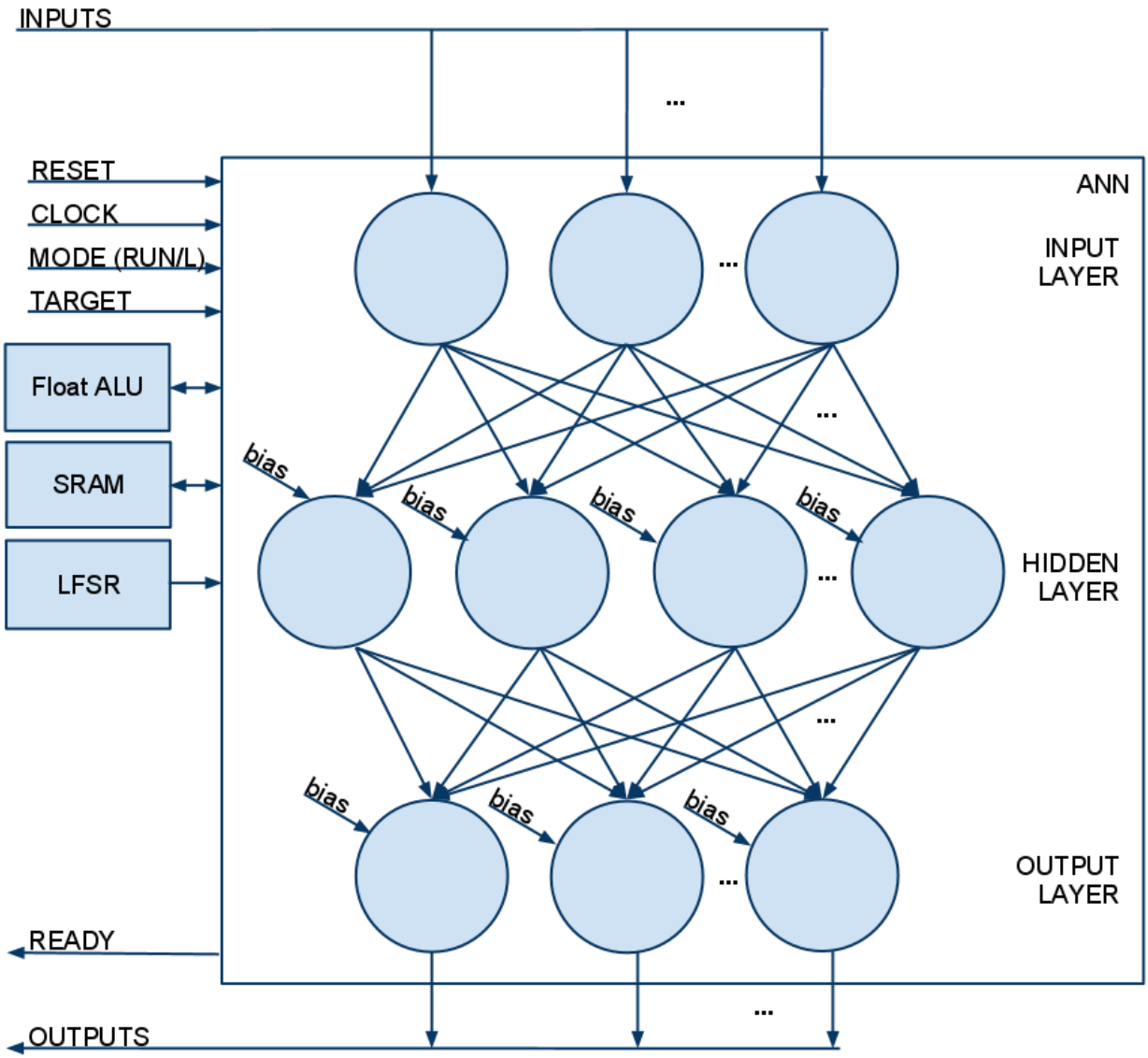

Figure 2: ANN design component block diagram

When the asynchronous RESET is high, the internal state machine of ANN is reset to the initial state. During the initialization phase, the ANN randomizes all connection weights. After the initialization phase, the ANN stays at idle state.

When mode is set to RUN, ANN makes the transition from idle state to run state, where it executes many other states sequentially to do the following:

1. For each perceptron in the hidden layer, calculate the weighted sum and the output from the sigmoid function
2. For each perceptron in the output layer, calculate the weighted sum and the output from the sigmoid function

Afterwards, ANN returns to idle state.

When mode is set to LEARN, ANN makes the transition from idle state to run state but with a learning flag set to HIGH. It executes many other states sequentially to do the following (the first two steps are identical to the RUN mode steps):

1. For each perceptron in the hidden layer, calculate the weighted sum and the output from the sigmoid function
2. For each perceptron in the output layer, calculate the weighted sum and the output from the sigmoid function
3. Calculate error for each perceptron in the output layer by subtracting output of such perceptron from the training target
4. Accumulate the error from output layer to compute Mean Squared Error (MSE)
5. Calculate delta as the derivative of the sigmoid function multiplied by the error for each perceptron in the output layer
6. Calculate delta weight for each of the input connections for each of the perceptrons in the output layer and updates the weight based on the delta weight for these connections
7. Calculate error for each perceptron in the hidden layer by using the deltas calculated for the output layer's perceptrons
8. Calculate delta as the derivative of the sigmoid function multiplied by the error for each perceptron in the hidden layer
9. Calculate delta weight for each of the input connections for each of the perceptrons in the hidden layer and updates the weight based on the delta weight for these connections

Afterwards, the ANN returns to idle state.

Implementation

In order to demonstrate the functionality of the ANN, a rudimentary example application is implemented on an Altera DE2 board to perform pattern recognition on a 4x4 binary grid.

User Interaction

The inputs into the Pattern Recognizer (PR) are the first 16 switches on the Altera DE2 board. Each group of 4 is one row of the grid. Due to difficulties in representing letters in the 4x4 grid only 16 letters are currently trained to be recognized: A, C, D, F, H, I, J, L, N, O, P, T, U, X, Y, Z. The following diagram shows the training data set of the pattern recognizer.

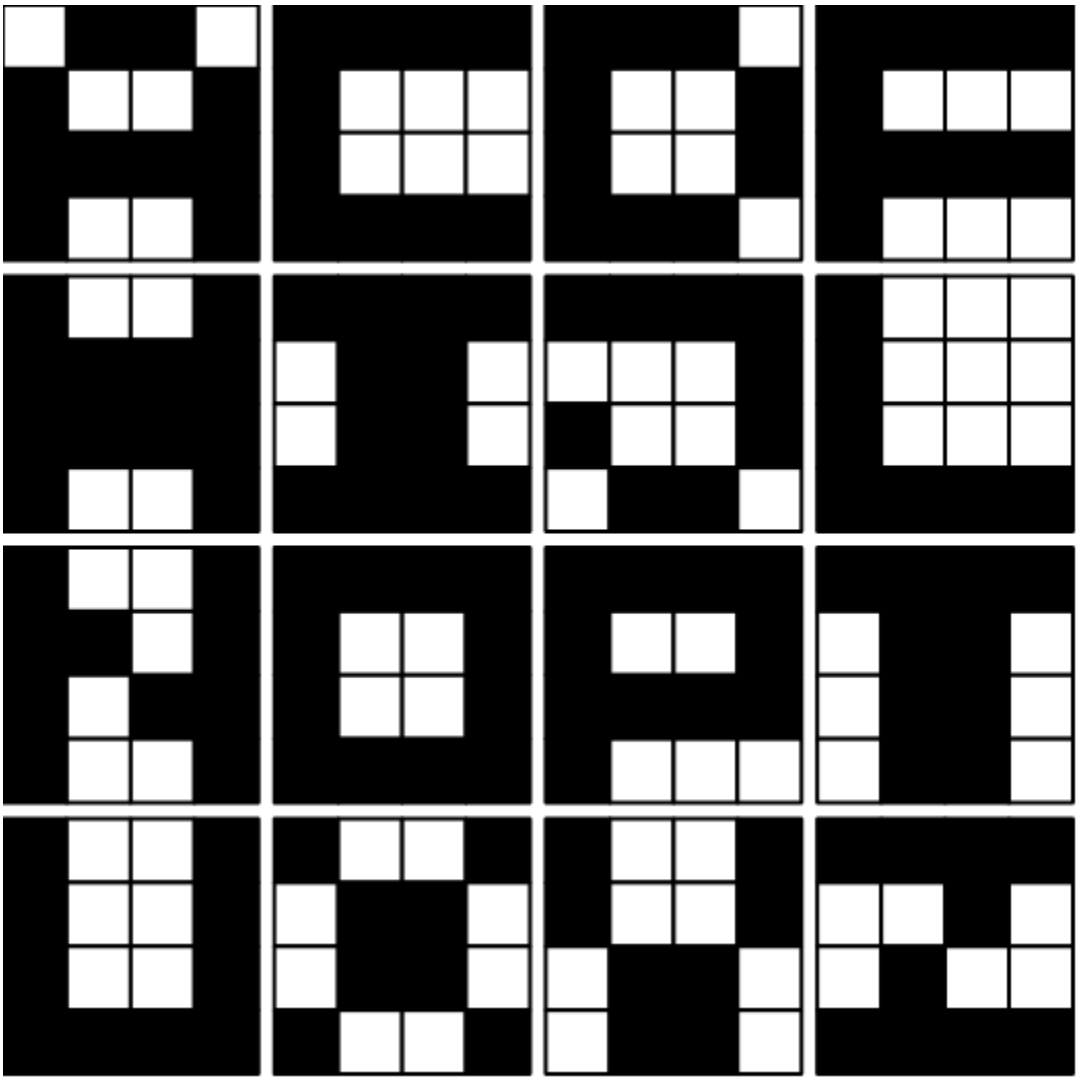

Figure 3: Pattern Recognizer training data set

To ease the user from guessing what pattern has been put in, the project uses the LCD display to show the current input pattern based on the states of the toggle switches.

Figure 4: LCD Display

Once the desired pattern is put in, the user needs to press the left most push button (KEY[3]) once to trigger the recognition process. At the end of the recognition process, the most probable classification of the input pattern is displayed on the LCD screen.

Pressing the right most push button (KEY[0]) will globally reset the FPGA, leading to a re-training of the pattern recognizer. Once the training is done, a green LED directly below the LCD is lit to indicate good condition and ready for user interaction.

Figure 5: LCD Display during Training

Pattern Recognition Network

The utilized ANN is composed of 16 perceptrons in the input layer, 32 in the hidden layer, and 16 in the output layer. Each of the 16 input layer perceptrons has an input value related to the state of the corresponding toggle switch on the 4x4 grid. A switch at HIGH position generates a value of 1.0 for the input of the corresponding perceptron. And a value of 0.0 will be used otherwise. The number of output perceptrons is corresponding to the number of patterns to be recognized. Each output perceptron will compute a value between 0.0 and 1.0 to indicate the likelihood of the pattern being in that particular class. For example, an output of 0.9 at output perceptron that corresponds to letter A means it is very likely that the input pattern resembles the letter A.

The ANN code is generic and these values are easily tweaked. This allows the ANN to be built for any number of inputs to the network as well as any number of outputs. For example, if a larger external board of switches was used the ANN could use more inputs to get a finer resolution on the grid. On the other hand, the ANN could be configured to have more outputs to recognize more letters and possibly numbers and symbols.

The ANN is implemented using IEEE single precision floating point. It requires a floating point ALU to perform any computation. Since there is no onboard built-in floating point ALU on the Altera DE2 board, one is synthesized from Mega-functions provided by Altera's IP were assembled into a floating point ALU which is used by the ANN. This ALU contains an adder, a subtractor, a multiplier, a divisor, an exponentiator and a comparator. Each ALU sub-component requires different clock cycle delays. However, ANN is designed to wait for variable number of cycles for each operation as long as the ALU provides a ready flag whenever operation is done.

The ANN is trained by the Pattern Recognizer (PR) component. This module handles the running and training of the ANN. If a different behavior is desired, only the PR need be swapped out with a new supervisor which will train the ANN for the new purpose. To begin training, the weights of the connections in the ANN are initialized to pseudo-random values through the use of a 16-bit linear feedback shift register (LFSR). The ANN outputs a mean squared error (MSE), which is a measure of how close its outputs were to the training set. When the PR determines that the MSE is below a threshold value for all its training data sets, it concludes that the training is complete

and signals ready for user input by lighting a green LED. This green LED is the indication of passing the Built-in Self-test (BIST). The ANN can be re-trained at any time in one of two ways: redo the initial training by resetting seed in the linear feedback shift register; or, retrain the LFSR seed and only resets the ANN which will result in a different set of initial weights and ultimately different ANN behavior.

The ANN uses the SRAM chip on the DE2 board to store the weights used in the ANN. When the weights were synthesized as registers on the FPGA, it resulted in about 95% usage of the logic elements. This didn't leave enough space for the controlling components to be synthesized. After offloading those to the SRAM, the usage dropped to around 10-15%. This allowed implementation of additional components which improve the user experience, such as the LCD display module.

Integration

The following block diagram highlights the outline of major components in the project's integrated design.

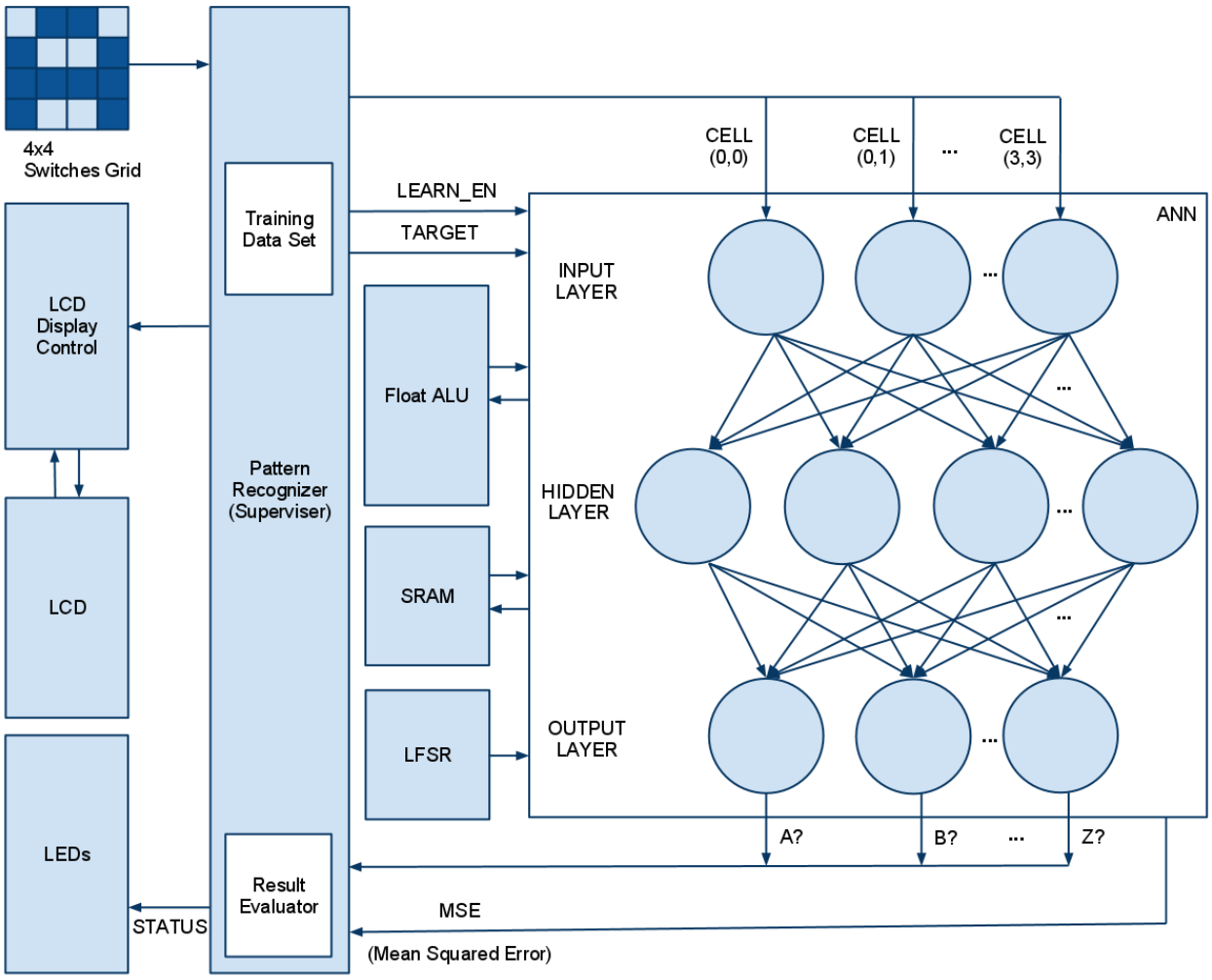

Figure 6: Block diagram of the integrated system

- **ANN**: generic artificial neural network with back propagation
- **Float ALU:** Contains IEEE 32-bit floating point add, sub, mul, div, exp, cmp
- **LFSR**: 16-bit pseudo random number generation
- **LCD**: LCD communication module
- **LCD Display** Control: logical control of display content
- **SRAM**: SRAM communication module
- **Pattern Recognizer**: supervisor for ANN to recognize patterns from the training data set

Challenges

Floating Point Implementation

Floating point is an essential part of a reliable artificial neural network. 32-bit single precision floating point allows ANN to pick up subtle features from the set of inputs that might otherwise be neglected. However, implementing floating point operations on FPGA is difficult mainly due to the amount of resources it requires.

Previous work on hardware ANN also faced this challenge. Attempts had been made to approximate inputs, outputs and connection weights using fixed point format. However, those implementations were usually severely limited by its precision leaving large room for error.

In this project, different floating point arithmetic unit packages had been experimented with, including IEEE proposed floating point implementation for Altera FPGAs, flopoco custom floating point arithmetic unit generator, and Mega-functions from Altera IP. The conclusion from these experiments showed that the Altera Mega-functions require the least number of logic elements and provide the fastest operations.

However, since floating point arithmetic units are still relatively expensive to use, only one of such ALU is implemented in the ANN. The massive computation of ANN had then to be executed in sequential order by the control of a finite state machine. This implementation tends to be much slower than those pipelined or massively parallel implantation of ANN using integer arithmetic.

Neural Network Size Limit

When implementing the ANN, many weights are needed to be stored to allow for a weighted sum of inputs at each perceptron. With each weights being a 32 bit floating point number, the number of registers used to store the weights of a moderate network was very substantial. The final product would have utilized close to 100% of the logic elements on the FPGA which would have made it much more prone to faults due to the layout. The solution to this issue was to utilize the SRAM present on the DE2 board. This allows the gate count to be greatly reduced while incurring a minimal cost of clock cycles to read and write. It adds 2 cycles to read, and 3 cycles to write to the SRAM.

Training Length

Through testing of the NIOS approach, it was determined that training time was greatly increased when the weights were all initialized to the same value. When the weights were initialized to different, random values the speed of convergence of the ANN increased dramatically. In the final implementation a linear feedback shift register is used to generate the pseudo-random numbers for the weights. This will result in a repeatable sequence of trained states for identical networks.

Another contribution factor to faster learning is the addition of bias connection on each of the perceptrons. The bias connection is same as any other connection except its input value is always set to 1. However, the back propagation algorithm will update the weight on this bias connection to provide a larger range of possible inputs to a perceptron and hence enabling the ANN to function more accurately.

Conclusion

This project required utilizing many features of the Altera DE2 board. These include the LCD display, the switch inputs, LEDs, and the SRAM. The switches and LEDs were straightforward to use in the design. The SRAM and LCD required looking at datasheets to determine how the controlling state machines would need to be designed to meet the timing requires of the units. The many stages of the implementation allowed the experiments of different technologies to implement parts of the system, and to determine which worked the best. The final system isn't the same as the proposed project. However it is equally if not more complex in the task that the ANN is trained to do.