**Executive Summary**

Our project was implementing an Artificial Neural Network (ANN) on a Field Programmable Gate Array (FPGA). Our desire was to use an FPGA’s concurrent nature to utilize the inherent parallel structure of an ANN. In order to successfully implement this ANN, we were required carefully design the ANN to fully take advantage of the concurrency of the FPGA.

**Objectives**

To design and implement an Artificial Neural Network that can efficiently execute on a Field Programmable Gate Array.

To become more familiar with FPGA design strategies

**Neural Network**

An Artificial Neural Network (ANN) is a common machine learning algorithm. They are typically used to recognize pattern and solve complex problems that are not well defined.

An ANN is made up of a number of “units” organized in layers. The units in each layer take in inputs from the previous layer, process the data, and then send the result to the units in the next layer. This progression of data is referred to as “Feed Forward”.

The data processing is accomplished by applying an activation function to the input data. There are many different possible activation functions and each has strengths and weaknesses.

The output of the ANN is, in essence, a guess of what the output should be. From here, the parameters of the network can be adjusted to bring the output closer to an expected training point through a process called back-propagation.

**Zynq**

For our project, we decided to use the Zybo Zynq 7010 FPGA. This board uses the Zynq7010 system on chip designed by Xilinx which combines an on-board processor and RAM with programmable logic. This allows for a hybridization of hardware and software to accomplish tasks.

To implement the ANN on this board, we used the Xilinx Vivado design suite. This program allowed us to design and test the components of our network using Verilog, a hardware description language as well as a number of modules designed by Xilinx which are included in Vivado.

**Design Strategy**

In order to design an ANN that would run efficiently on an FPGA, we had to find ways to reduce the size and complexity of the modules involved.

Floating point arithmetic was replaced by integer arithmetic to reduce complexity, size and time required for floating point operations.

Multi-cycle multiplication was replaced by bitwise shifts to reduce operation time to close to one clock cycle.

Complicated exponential activation functions such as Gaussian or Sigmoid which do not map well to hardware were replaced with an Elliot function which only requires a single division operation.

Multiple layers of dedicated units were substituted with layer multiplexing, where a single unit had the capability of performing the functions of a single unit in each layer.

While many of these changes successfully improved the space and time efficiency of our ANN, they also limited the accuracy of our results. However, the saved space would allow for more units to be added to the network which would theoretically compensate for this loss.

**Implementation**

**Results**

We have successfully designed an ANN that can be easily mapped to an FPGA. Our designs do not rely on specialized on-board logic and could easily be implemented on a wide range of FPGAs or integrated circuits.

We were able to successfully simulate execution of the units that control the ANN operations as seen in Figure ?.

//control waveforms!

**Future work**

The back-propagation algorithm still needs to be derived and implemented on the Zynq processor.

Timing analysis needs to be completed to analyze performance in comparison to a widely used serial software approach.

Cost analysis needs to be completed to analyze the potential market value of a similar design.