TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol. 14, No. 1, April 2015, pp. 130 ~ 139
DOI: 10.11591/telkomnika.v14i1.7272

Hardware Implementation of FIR Neural Network for Applications in Time Series Data Prediction

Kuldeep S. Rawat[1], G. H. Massiha*[2]
[1]Department of Technology, Elizabeth City State University, North Carolina, USA 27909
[2]Department of Industrial Technology, University of Louisiana at Lafayette, Louisiana 70504
*Corresponding author, e-mail: ksrawat@mail.ecsu.edu[1], massiha@louisiana.edu[2]

Abstract

Time series data prediction is used in several applications in the area of science and engineering. Time series prediction models have been implemented using statistical approaches, but recently, neural networks are being applied for times series prediction due to their inherent properties and capabilities. A variation of a standard neural network called as finite impulse response (FIR) neural network has proven to be highly successful in achieving higher degree of prediction accuracy when used over various time series prediction applications. These applications are time critical and involve huge amounts of computation that are slower when run on a general purpose processor and hence, a dedicated hardware is required. In this paper, authors present hardware implementation of an FIR neural network for applications in times series data prediction. The implementation is divided into (i) off-board, where the training algorithm and neural network configuration is implemented in Matrix Laboratory (MATLAB) and simulated with various benchmark time series data set and (ii) on-board, where the entire system is modeled in a hardware description language (HDL). The simulation experiment, hardware building blocks, the implementation framework, and the hardware design flow are discussed in this paper. The hardware resource utilization and timing information are also reported in the paper.

Keywords*: neural network, FPGA, time-series data, HDL, electronics, rapid prototyping*

1. Introduction

Time series is a set of observations generated sequentially in time. Often the problem of interest in these observations is the prediction of some future values based on the recent past. This problem of time series prediction has applications in such areas as hydrology, transportation, telecommunications, quality control, financial world, and industrial processes [1]. The goal of time series prediction is to find a function $f:\Re^N \to \Re$ to obtain an estimate of *x* at time $t+d$, so that:

$$x(t+d) = f(x(t), x(t-1), ..., x(t-N+1)) \quad (1)$$

$$x(t+d) = f(y(t)), \quad (2)$$

Where $y(t)$ is the *N*-ary vector of lagged x values.

Artificial neural networks (ANNs) are general function approximators that have been applied in pattern recognition, classification, and process control [1, 2]. Recently, they are being used in the areas of prediction, where regression and other related statistical techniques have traditionally been used. One class of ANNs is the feedforward networks, which has one or more hidden layers of neurons, also referred to as hidden units [2]. The nonlinearities of the hidden units allow the network to extract higher-order statistics and are particularly valuable when the size of the input layer is large. The architectural graph shown in Figure 1 illustrates the layout of a multilayer feedforward neural network. For brevity, the network in Figure 1 is referred to as a 4-3-3-4 network in that it has 4 source nodes, 2 hidden layers of 3 hidden neurons, and 4 output neurons.

Received January 1, 2015; Revised March 10, 2015; Accepted March 25, 2015

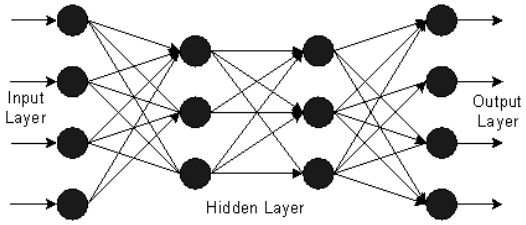

Figure 1. Fully connected feedforward network

The standard neural network method of performing time series prediction is to approximate the function f in neural network architecture, using a set of *N*-tuples (finite sequence of data points) as input and a single output as the target value of the network. This method is often called the sliding window techniques as the *N*-tuples input slides over the training set [2]. The basic architecture of a sliding window method for time series prediction is shown in Figure 2.

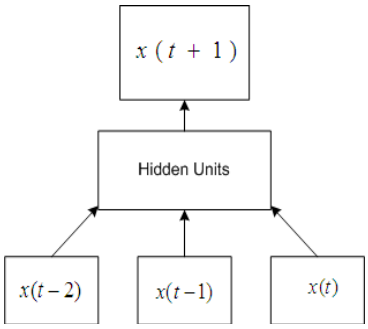

Figure 2. Sliding window based time series predictor

Time series data predictor can be either implemented in software or hardware. Software solutions are comparatively slower as large numbers of computations are involved. Several researchers have adopted hardware implementation with great success [3, 4] (Restrepo, Hoffmann, Perez-Uribe, Tuescher & Sanchez, 2000; Beiu, 1996). These hardware implementations facilitate the use of neural network in real-time applications. These real-time applications range from predicting voice traffic demand, temperature prediction of a blast furnace, and prediction of product quality in a chemical process, to rainfall-runoff modeling [1]. These applications involve huge amounts of computation that are slower when run on a general-purpose processor [5]. Using dedicated hardware, one can achieve higher speed and real-time analysis of data and prediction results. Recently, reconfigurable hardware solutions in the form of FPGAs offer high performance with the ability to be electrically reprogrammed to perform changes in design and algorithm [6-8].

This work focuses on FPGA implementation of finite impulse response (FIR) neural network for time series data prediction. The first step involves simulation of temporal backpropagation training algorithm using Matrix Laboratory (MATLAB). The outcome of MATLAB simulation is the design parameters or the FIR neural network topology for the best prediction. This information is later used for hardware implementation, where design is modeled in hardware description language (HDL).

The rest of the paper is organized as follows. The next section presents FIR neural network model. Section 3 presents in detail the building blocks of FIR neural network. In section 4 authors discuss the hardware implementation framework. Both simulation experiment and FPGA implementation are discussed in this section. Finally relevant conclusion is presented.

2. FIR Neural Network

Finite impulse response (FIR) neural network, first proposed by Eric Wan, had great success at the Santa Fe Institute (SFI) time series prediction competition [9, 10]. The FIR neural network design outperformed other competitors in the prediction tasks. In an FIR neural network each neuron is extended to be able to process temporal features by replacing each synaptic weight by an FIR filter [2, 10]. An FIR neuron model is shown in Figure 3, with corresponding time delay neural network representation shown in Figure 4.

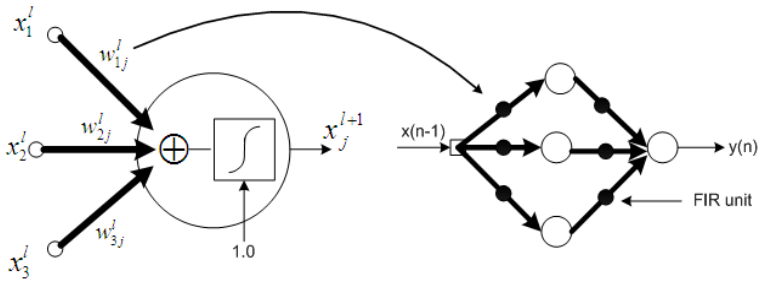

Figure 3. FIR neuron model

An FIR neural network's input layer consists of FIR filters, feeding the data into neurons in hidden layer. Similar to conventional feedforward networks, an FIR neural network may have one or several hidden layers. The output layer consists of FIR neurons that receive their inputs from previous hidden layer. The network shown in Figure 4 consists of three layers with a single output neuron and two neurons in hidden layer.

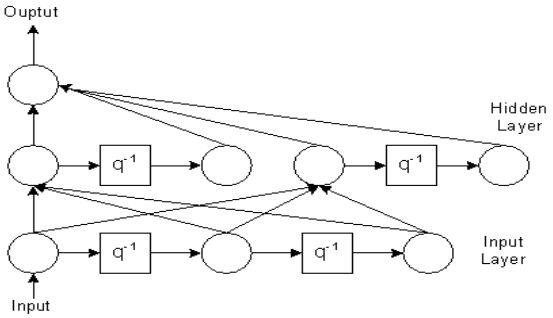

Figure 4. A time delay neural representation of FIR neural net

As seen in Figure 4, all the connections are delayed (time processed) before passing on to neurons in the next layer. In effect, the network is unable to learn temporal features that are longer than its filter lengths summed together. Consequently, selection of the lengths of FIR filters is quite critical in achieving good prediction performance.

3. Building Blocks of FIR Neural Network

The basic computing modules used in the FIR neural network design are, addition, multiplication, and unit delay. Such circuits as the adder tree, multiplier-accumulator unit, and the FIR filter can be designed using these basic modules. In this work, authors used digit-serial architecture that combines the area efficiency of a bit-serial architecture with the time efficiency of bit-parallel architecture [11, 12]. In the digit-serial approach, data words are divided into digits, having a digit size *N*, which are processed in one clock cycle. Architectures based on the digit-serial approach offers a better overall solution considering the tradeoffs between speed, efficient area utilization, throughput, I/O pin limitations and power consumption. The digit-serial approach also leads to a regular layout and the possibility of building a pipeline with it. A brief

description of digit-serial architecture of each computational unit used in FIR neural network implementation is as follows.

3.1. Digit-serial Adder and Digit-serial Multiplier

A basic element in a digit-serial arithmetic implementation is the digit-serial adder shown in Figure 5(a). The two operands, *A* and *B*, are fed one digit at a time into the digit-serial adder. The addition is done *N* bits at a time, with the carry rippling from one full adder to the next. The carryout from the digit-serial adder is fed back into the first full adder during the next clock cycle, when the next digits of the inputs have arrived. The digit-serial adder is further used to form digit-serial multiplier.

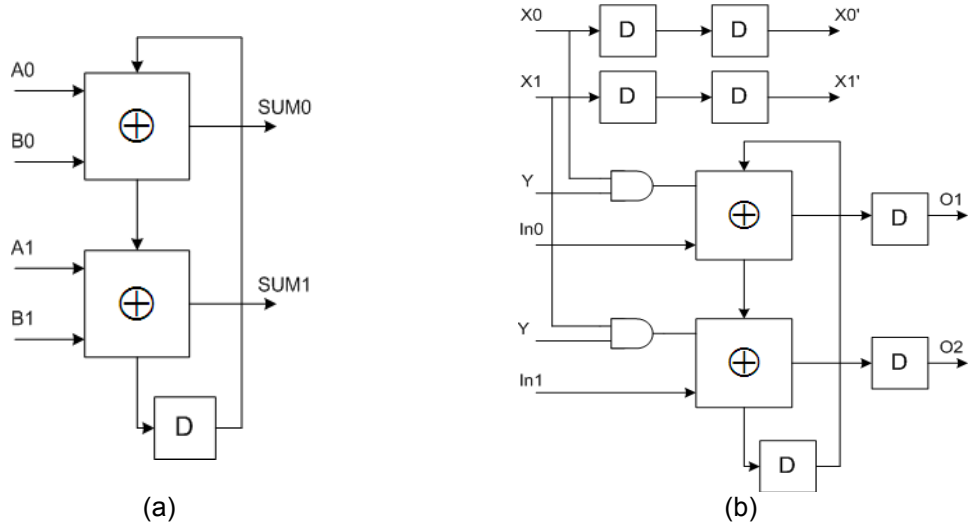

Figure 5. a) A digit-serial adder module. b) Digit-serial multiplier module

Multipliers are used in multiplying weights values to the incoming times series data. The simplest way is add-shift multiplication as shown in Figure 5(b). The *N*-bit operands for multiplication are stored in two registers and multiplied with each other in *N* steps. A 2-input AND gate generates each partial product.

3.2. Digit-serial Pipelined Multiplier

In order to increase the throughput of digit-serial multiplier, each digit-serial multiplier module (DSMM) is connected in a systolic array fashion to implement a very fine-grained pipeline [13]. The bits of the multiplier are supplied one digit at a time, starting with the least significant digit, while the bits of the multiplicand are supplied as a parallel word. Each partial product is shifted and then added to the previous partial products. Figure 6 shows a digit-serial pipelined multiplier connected in systolic array fashion. Pipelining is done in order to limit the critical path propagation delays between registers. In the digit-serial pipelined multiplier shown in Figure 6, the pipelining limits the propagation to a 2-bit adder in the digit-serial multiplier with *N=2*.

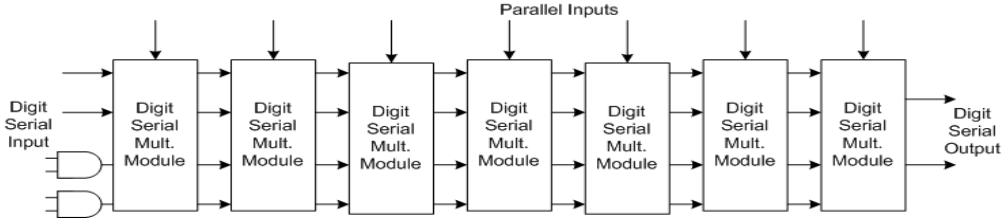

Figure 6. Digit-serial pipelined multiplier

3.3. Multiplier-accumulator Unit

In multiply-accumulate (MAC) operation the synaptic filter coefficients (weights) are multiplied with time series data and the results are added to an accumulator. Figure 7 shows the pipelined structure of multiply-accumulate unit. MAC unit consists of a multiplier followed by an adder and an accumulator register which stores the result when clocked. The output of the register is fed back to one input of the adder, so that on each clock cycle the output of the multiplier is added to the register.

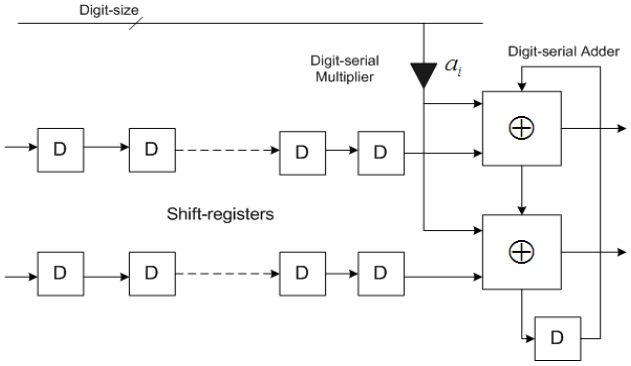

Figure 7. Multiply-accumulate unit

3.4. Digit-serial FIR Filter Design

As discussed earlier, in the case of an FIR neural network, a finite impulse response filter replaces the synaptic weights [10]. An FIR filter can be implemented using just three digital hardware elements, a unit delay (a latch), a multiplier, and an adder. Figure 8 shows a tapped delay line implementation of an FIR filter. The unit delay simply updates its output once per sample period, using the value of the input as its new output value. An FIR filter can be represented as the convolution sum as given in Equation (3). Notice that at each *k* we have access to the *M*+1 samples $x(k), x(k-1), x(k-2), ..., x(k-M)$.

$$y(k) = \sum_{m=0}^{M} h(m)x(k-m) \tag{3}$$

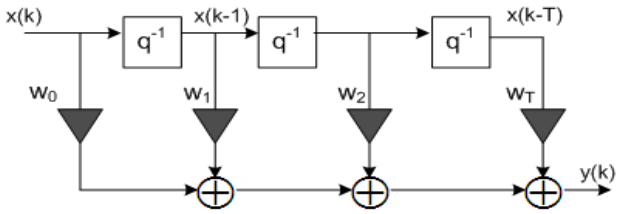

Figure 8. Tapped delay line implementation of FIR filter

3.5. Sigmoid Generator

The activation function or squashing function limits the output of neuron. The activation function used in this implementation is a sigmoid function. Sigmoid function was chosen as it is differentiable, which is an important property for applying learning algorithm. A sigmoid function is defined in Equation (4)

.

$$f(x) = \frac{1}{1+e^{-\lambda x}} \tag{4}$$

Where λ is the slope parameter of the sigmoid function. This function is a nonlinear function of *x* and is limited between the values of 0 or 1. The sigmoid function was implemented as a lookup tablet (LT). The LT implementation has a major advantage in that it does not require special hardware. Other implementation such as piecewise approximation is also a good choice for an inexpensive hardware implementation, but is more suitable for hardwired design [14].

3.6. The Architecture of a Neuron

The most important part of a neuron is the pipelined multiplier, which performs high-speed multiplication of synaptic signals (time series data) with filter weights as shown in Figure 9. An 8-bit data format is chosen for synaptic signals as well as for the weights. As seen in Figure 9, each neuron has a local read only memory (ROM) that stores as many coefficient values as there are connections (filter taps) to the previous layer.

An eighteen-bit accumulator is used to add the signals from the pipeline with the neuron's bias value, which is stored in a separate register. The output of the accumulator register is buffered before passing on to sigmoid generator. A register holds the neuron's computing result until it is ready to write the value to a shared output bus. All neurons of a layer use the same bus to address the sigmoid generator. The sigmoid function is programmed as lookup table that is downloaded with the bitstream file during FPGA synthesis.

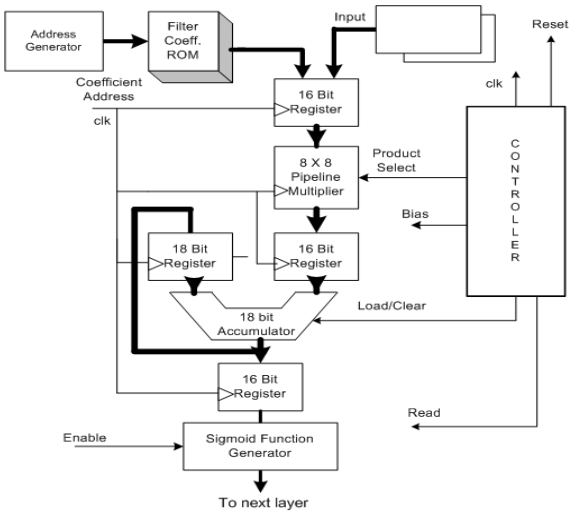

Figure 9. Architecture of neuron with its input and output signals

In Figure 9, the controller block generates a proper sequence of signals to control the timing for nodes in each layer. It generates addresses for the coefficient ROM and controls the time of multiplication, accumulation, using the output bus and preloading the bias, and enable and read sigmoid function LT. The controller is modified according to the number of neurons that want to use the common bus.

In a layered feedforward network, not all the neurons have to be connected together; rather, data are passed from one layer to the next. Every neuron has its multiplier, and all neurons may receive one data word in parallel, feed it to a pipeline multiplier, accumulate the multiplication results and write them to the common bus leading to the next layer one after each other.

4. Hardware Implementation Framework for FIR Neural Network

When implemented in hardware, neural networks can take full advantage of their inherent parallelism and run orders of magnitude faster than software implementations [3, 6, 15]. The implementation of FIR neural network is divided into two parts, off-board simulation and on-board hardware implementation, as shown in Figure 10.

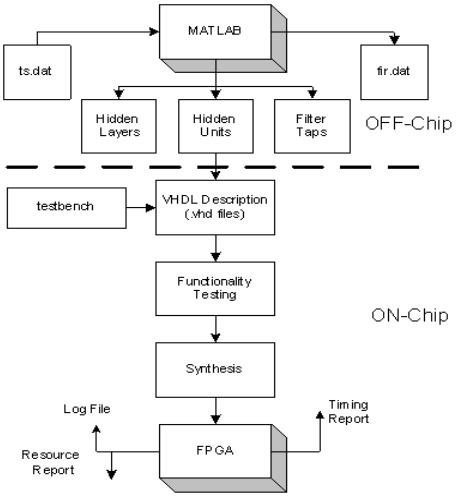

Figure 10. A complete implementation framework for FIR neural network

4.1. Off-board Simulation Experiment

The first part is an off-board simulation, where the temporal backpropagation learning algorithm is implemented in MATLAB. Details on the temporal backpropagation algorithm can be found in [2], [9-10]. The simulation program reads a time series data file, with the network configuration supplied by the user. The time series data sets used for the experiments were the benchmark sets of load in electrical net and fluctuations in a far-infrared laser. In addition, one set of synthetically generated time series was also used. This time series was generated by numerically integrating the equation of motion for a damped, driven particle in the potential field. The equation of motion was integrated with a simple fixed-step 4^{th} order Runge-Kutta routine that generated 100,000 data points.

Table 1 shows the number of data samples used to train networks and to test network generalization ability for each time series. Training data starts from the beginning of the series and test data starts from the end of the training data. The normalized mean square error (NMSE) given in Equation (5) was used as a performance measure for prediction accuracy. The idea is to minimize the mean square error (MSE) so as to achieve better prediction results. In Equation (5), σ^2 is the variance of the desired outputs d_i and N is the number of patterns.

$$NMSE = \frac{1}{N\sigma^2}\sum_{i=1}^{N}(x_i - d_i)^2 \quad (5)$$

Table 1. Training and test data set sizes

Time Series	Load in electrical net	Fluctuations in a far-infrared laser2	Synthetically generated time series
Training Set	1500	1000	5000
Test Set	500	1000	5000

The FIR neural network configuration that was used during simulation had two hidden layer and one nonlinear output neuron. Different combinations of prediction order and number of neurons in hidden layer were tried in effort to find the configuration that would model the data most effectively. The outputs of off-board learning are filter coefficients and the design parameters (number of hidden layers, number of hidden units, number of filter taps) of the best possible FIR neural network topology for the times series data input. This information is later used for hardware implementation. The final implemented architecture was based on the FIR neural network topology obtained for synthetically generated time series. The final architecture FIR neural network configuration obtained was a 1:10:10:1 fully connected feedforward network with the topological description shown in Table 2.

Table 2. Topology of the final FIR Neural Network

Topology	# of taps per synapse	Training set error	Test set error
1:10:10:1	20:4:4	0.5683	0.5838

4.2. On-board Hardware Implementation

Equipped with design parameter values obtained from off-board simulation, one can proceed to the on-board hardware implementation. The target device used for implementation was XILINX XC4000 series FPGA. Figure 11 shows XILINX XC4000 series FPGA development board [16, 17].

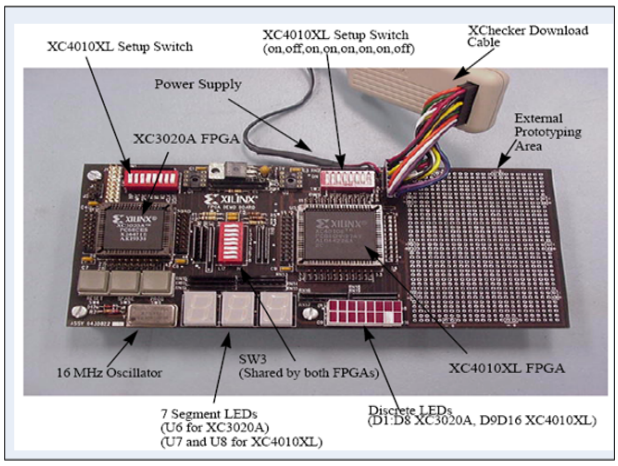

Figure 11. XILINX XC4000 series FPGA development board

The XILINX XC4000 series FPGA architecture is shown in Figure 12. It consists an array of programmable function units called Configurable Logic Blocks (CLBs), linked by programmable interconnect resources. The internal signal lines interface to the package through programmable Input/Output Blocks (IOBs). FPGAs are configured by setting each of their programmable elements (i.e. logic cells, routing network and I/O cells) to a desired state. These are programmed via programmable switches (PSM).

The XILINX XC4000 series FPGA used for the implementation has 576 CLBs in form of a 24 x 24 matrix. In addition to this it has 1,536 flip-flops and 192 IOBs. Each CLB consists of function generators, flip-flops, SRAM, and fast carry logic for addition [16, 17].

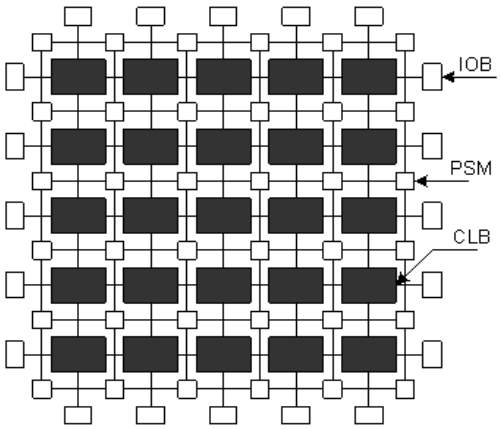

Figure 12. XILINX XC4000 series architecture

The hardware implementation involves modeling each computation module in a hardware description language (HDL). In this implementation, Very-High-Speed-Integrated-Circuit HDL (VHDL) was used to model hardware building blocks and the functionality was tested in ModelSIM [18]. All the modules were then integrated to form the complete FIR neural network system. The VHDL description of the final topology was synthesized on to FPGA using XILINX synthesis tool (XST). The FPGA design flow is summarized in Figure 13.

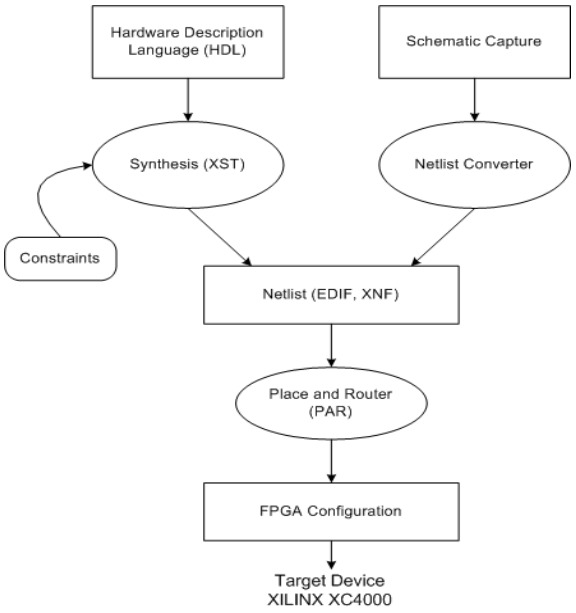

Figure 13. FPGA synthesis steps

The hardware description when run through XST generates a netlist of the hardware design in XILINX netlist format (XNF). This netlist file describes the connectivity of FIR neural network design and is passed through XILINX Placement and Routing (PAR) tool for efficient design layout. The output of a placement and routing operation is a bitstream file that contains all the FPGA configuration information. This bitsream is subsequently downloaded on to XILINX XC4000 series FPGAs. The resource utilization for the final FIR neural network design as reported by XST is summarized in Table 3. The maximum critical path delay was reported as 220nsec. Thus, FIR neural network board can be operated at maximum speed of 4.54MHz (1/220nsec). The implemented FIR neural network hardware can work with a host PC as a dedicated coprocessor, offloading the majority of the computation.

Table 3. Hardware resource utilization for FIR neural network

Resources	Number of units used
Configurable Logic Blocks (CLBs)	2204
Input/Output Blocks (IOBs)	182
Flip-flops or Latches	3862

5. Conclusion

This paper investigated finite impulse response (FIR) neural network design and its hardware implementation for time series prediction. The network incorporates FIR filters, which gives dynamic connectivity to the network and facilitates temporal processing of signal propagation in the synapses. To demonstrate potential applications, three different times series data sets were selected. One of the data set, synthetically generated, was used to obtain the final FIR neural network topology for hardware implementation. A complete design framework

for implementing FIR neural network on FPGA was presented. The training algorithm was implemented off-board using MATLAB.

On-board or hardware implementation procedure was in compliance with the FPGA design flow. The complete VHDL description was synthesized on to XILINX XC4000 series FPGA using XILINX synthesizing tool (XST). The synthesis tool also reported the resource utilization and maximum critical path delay. The final FIR neural network design can be operated at a maximum frequency of 4.54 MHz. Reconfigurability, adaptability, and scalability are the main features of this FIR neural network hardware design. For a new application, only the filter coefficients and biases need to be reconfigured on the FPGA without changing the basic design. The design can be easily expanded by just adding more nodes with the same configuration. The hardware can work independently or act as a coprocessor with a host computer running off-board temporal backpropagation algorithm.

References

[1] Widrow B, Rumelhart D, Lehr M. Neural networks: Applications in industry, business, and science. *Communications of the ACM*. 1994; 37: 93-105.
[2] Haykin S. Neural Networks: A Comprehensive Foundation. 2nd Ed. Upper Saddle, NJ: Prentice Hall. 1999.
[3] Restrepo HF, Hoffmann R, Perez-Uribe A, Teuscher C, Sanchez E. *A networked FPGA-based hardware implementation of a neural network application.* In Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines. 2000: 337-338.
[4] Beiu V. Optimal VLSI implementations of neural networks: VLSI-friendly learning algorithms. Neural Networks and Their Applications. J. G. Taylor. Chichester, UK: John Wiley. 1996: 255-276.
[5] Waldeck P, Bergmann N. *Evaluating software and hardware implementations of signal-processing tasks in an FPGA.* In Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology. 2004: 299–302.
[6] Jung S, Kim S. Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems. *IEEE Transactions on Industrial Electronics*. 2007; 54(1): 265-271.
[7] Jihong L, Deqin L. *A survey of FPGA-based hardware implementation of ANNs*. In Proceedings of the 2005 International Conference on Neural Networks and Brain, ICNN&B. 2005; 2: 915-918.
[8] Zhu JM, Gunther BK. *Towards an FPGA based reconfigurable computing environment for neural network implementations*. In Proceedings of the 9th International Conference on Artificial Neural Networks. 1999; 2: 661-666.
[9] Wan E. *Temporal backpropagation for FIR neural networks.* In Proceeding of the International Joint Conference on Neural Networks (IJCNN). 1990; 1: 575-580.
[10] Wan E. Finite Impulse Response Neural Networks with Applications in Time Series Prediction. PhD Dissertation. Stanford University; 1993.
[11] Aggoun A, Ibrahim MK, Ashur A. Bit-level pipelined digit-serial array processors. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing.* 1998; 45(7): 857-868.
[12] Artley RI, Parhi KK. Digit-serial Computation. Boston, MA: Kluwer Academic. 1995.
[13] Kim CH, Kwon S, Hong CP. *A fast digit-serial systolic multiplier for finite field GF(2/sup m/).* In Proceedings of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC'05). 2005; 2: 1268-1271.
[14] Zhang M, Vassiliadis S, Delgado-Frias JG. Sigmoid generators for neural computing using piecewise approximations. *IEEE Transactions on Computers*. 2002; 45: 1045-1049.
[15] Rafael G Cerdá J, Ballestar F, Mocholi A. *Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation.* Proceeding ISSS '00 Proceedings of the 13th international symposium on System synthesis. 2000: 225-230.
[16] XILINX. The Programmable Logic Data Book. Xilinx, Inc. 1996.
[17] Parnell K, Mehta N. Programmable Logic Design Quick Start Handbook. Xilinx, Inc. 2003.
[18] Ashenden PJ. The Designer's Guide to VHDL (Systems on Silicon). 2nd Ed. San Francisco, CA: Morgan Kaufmann Publisher. 2002.