HW/SW Codesign of FPGA-based Neural Networks

Alper Ucar and Ali Ziya Alkar

Hacettepe University,
Department of Electrical & Electronics Engineering,
06800 Ankara, Turkey
{ucar,alkar}@hacettepe.edu.tr

Abstract. In this article, we present a HW/SW codesign approach for the implementation of multilayer perceptrons resulting in an embedded system that can be used in wide variety of applications. The motivation for the HW/SW codesign includes declining time-to-market and power constraints as well as increasing gap between silicon area and computational intensity. By utilizing codesign, hardware tasks –to be implemented either on ASIC or reconfigurable logic device– can be organized in a way that is compatible with the software tasks running on a host computer or a DSP. In our model a general-purpose computing platform acts as the master controller which transmits appropriate signals to an FPGA-based coprocessor. An array of processing elements (PEs) is mapped onto a single FPGA for forward propagation to exploit the parallel nature of the network architecture. Synthesis results indicate high speed operation with limited number of PEs which may be a significant contribution in designs where high throughputs can be obtained in low cost FPGA's.

1 Introduction

Artificial neural networks (ANNs) consist of massively parallel interconnected simple processors (neurons) intended to provide solutions in the area of pattern recognition, system identification, and time-series forecasting. One of the most widely used network model is the multilayer perceptron (MLP) with error backpropagation learning. Backpropagation learning algorithm is a computationally intensive task and software implementations suffer from large execution times which make them inadequate in meeting the demands of real-time processing applications.

The inherent parallelism of ANNs is well-suited for hardware implementations. The high-degree of parallelism can be exploited by mapping an array of PEs to the desired structure. PEs are arithmetic units such as multiply and accumulate circuit (MAC) for matrix-vector multiplication or nonlinear processing unit for realizing the activation function. Learning is accomplished by updating the weight matrix in order to minimize the error function.

The advent of rapid prototyping tools facilitated implementation of neural architectures. There have been several studies [1-4] implementing the FPGA-based ap-

proaches for the ANNs. A major bottleneck has been the limited logic density of FPGAs to implement backward propagation phase of the algorithm as well as lack of coherent techniques to resolve suitable network topology. An overall low-cost FPGA implementation regarding every single phase of backpropagation learning is not feasible considering the issues stated above.

The HW/SW codesign approach can be considered as an embedded computing application where the hardware and software must be designed together to make sure that the implementation not only functions properly but also meets performance, cost, and reliability goals [5]. This approach is also emerging as an efficient method for the design of neural and neuro-fuzzy systems. Recent applications include Hopfield type network [6] to solve the task scheduling problems in embedded and real-time systems, neural controller design [7], and neuro-fuzzy hardware design [8].

The rest of the paper is organized as follows: Section 2 briefly introduces the problem formulation of MLPs on FPGAs. Section 3 proposes the embedded design environment. The synthesis results are presented in Section 4. Finally, Section 5 is the conclusions.

2 Problem Formulation

2.1 Backpropagation Learning

The steps of the backpropagation learning for MLPs are multiplication-rich and can be separated into three distinct phases: the recall (forward propagation) phase, where the outputs of the neurons due to an input pattern $[\bar{x}(1),\mathrm{K},\bar{x}(n)]$ are calculated; the learning (backward propagation) phase, where the error terms of the neurons in the output and hidden layers are determined; and the weight adaptation phase, where the synaptic weights are updated to minimize the error function. Phases of the algorithm for a MLP with one hidden layer are presented in Fig1. In Fig.1, σ represents the nonlinear activation function such as the sigmoid, $[\bar{d}(1),\mathrm{K},\bar{d}(n)]$ is the desired response of the system, and $0<\eta<1$ is the learning rate of the network.

2.2 Target Platform

Target platform Xilinx XCV600e FPGA uses 0.18 μm CMOS process and contains two major configurable elements: configurable logic blocks (CLBs) and I/O blocks (IOBs). CLBs provide the functional elements for constructing logic while IOBs provide the interface between the package pins and the CLBs. The architecture of a CLB contains four logic cells and is organized in two similar slices. Each logic cell (LC) includes a 4-input look-up table (LUT), dedicated fast carry-lookahead logic for arithmetic functions, and a flip-flop [10].

1 Calculate the outputs of the neurons in the hidden and output layer.

$$y_j(p) = \sigma\left[\sum_{k=0}^{K} w_{jk} x_k(p)\right] = \sigma(net_j),\ 1 \le j \le J, \tag{1a}$$

$$y_i(p) = \sigma\left[\sum_{j=0}^{J} w_{ij} y_j(p)\right] = \sigma(net_i),\ 1 \le i \le I, \tag{1b}$$

2 Calculate the error terms of the neurons starting from the output layer and moving backwards to the hidden layer.

$$\delta_i(p) = \sigma'\left(net_i(p)\right)\left[d_i(p) - y_i(p)\right], \tag{2b}$$

$$\delta_j(p) = \sigma'\left(net_j(p)\right)\sum_{i=1}^{I} w_{ij}\delta_i(p). \tag{2b}$$

3 Update synaptic weights.

$$\Delta w_{ij} = \eta \delta_i(p) y_j(p), \tag{3a}$$

$$\Delta w_{jk} = \eta \delta_j(p) x_k(p). \tag{3b}$$

Fig.1. Phases of backpropagation learning

2.3 Arithmetic Representation

Selection of the weight precision is a critical issue when implementing ANNs on FPGAs. While higher weight precision results in fewer quantization errors, lower precision has the advantage of greater speed and reduction in area. In order to resolve the trade-off, Holt and Baker [11] investigated the minimum precision required for a class of benchmark classification problems and concluded that 16-bit fixed-point representation is considered to be an optimal precision vs. area trade-off for FPGA based ANNs. Fixed-point representation has limited range compared to that of floating point, but it has the advantage of being as fast as integer arithmetic. Table 1 illustrates the data representations with respect to the components in this study.

Table 1. Data representations used

Component	Range	Length S	Length I	Length F	Representation
Inputs	[-1.0,1.0]	1	-	15	Signed fixed-point
Outputs	[0.0,1.0]	-	-	16	Unsigned fixed-point
Synaptic weights	[-8.0, 7.9998]	1	3	12	Signed fixed-point
Activation function	[0.0,1.0]	-	-	16	Unsigned fixed-point

3 HW/SW Codesign Environment

3.1 Phases of the Design

An embedded system design is formed by applying four major transformations [5]:

- Partitioning the algorithm to be implemented into smaller pieces,
- Allocating those partitions to the microprocessors and hardware units,
- Scheduling the times at which functions are executed,
- Mapping the generic algorithm description into an implementation on a particular set of components, either as software suitable for a given microprocessor or a logic device which can be implemented from the given hardware libraries.

The suggested system is partitioned into individual modules. A task manager is running on the host computer acts as the master controller. Training executed on the software unit to determine the synaptic weights that satisfy the convergence criterion for a specified MLP architecture. UART module, with 115200-baud rate, transmits the appropriate synaptic weights to the FPGA coprocessor.

SRAM on the coprocessor is employed to store the synaptic weights received form the host computer. Once the software unit determines the appropriate synaptic weights, the feed-forward operation is executed on the PEs to carry out the desired response of the network. A finite state machine (FSM) is implemented on FPGA for the synchronization of the feed-forward propagation. The architecture proposed, avoiding on-chip backpropagation learning, allows high throughput with low area cost. The overall design is shown in Fig. 3a.

3.2 Software Partition

A software device driver, implemented in C, is responsible the following tasks:

- Training. Network is trained for a given MLP architecture. The training data is presented as ASCII in a text file.
- Initialization of the FPGA coprocessor. Once the training is complete, the main program transmits the MLP configuration and the synaptic weights to the coprocessor through the RS232 communications interface.
- Monitor run-time progress. The main program displays the run-time data generated by the coprocessor to the end-user.
- Obtain the output data. The main program retrieves the coprocessor output and displays it to the end-user.

3.3 Hardware for MLP

The stages of the backpropagation algorithm can be expressed as basic matrix operations which enable mapping the structure onto parallel architectures such as systolic arrays [12]. To imitate recall phase of the backpropagation, a ring systolic array is constructed where each PE comprises a pipelined fixed-point multiply and accumulate circuit (MAC) accompanied by a sigmoid approximator as illustrated in Fig.2b and Fig.2c.

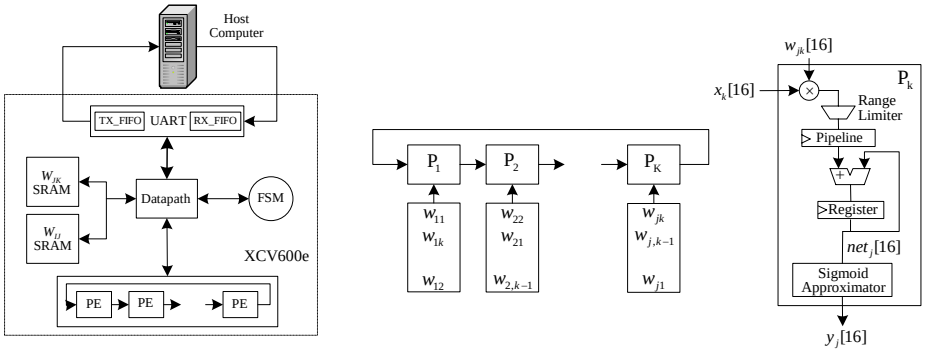

Fig. 2. **(a)** Architecture of the system **(b)** Ring array representation for MLP **(c)** Internal datapath of the PE

Input vector $\bar{x}(p)$ and synaptic weights are shifted at each clock cycle by horizontal and vertical shifters and a partial sum is calculated in every PE. Assuming an input vector of length K is connected to J neurons in the hidden layer and network has I neurons in the output layer, the weighted sum –referred as the net– is calculated in $J+I$ cycles with a K-processor array for the hidden layer and a J-processor array for the output layer. When accumulation stage is pipelined, two more clock cycles are required to obtain the net.

Activation functions are needed to introduce nonlinearity into the network. High-speed computation of sigmoid activation function can be performed by piecewise linear approximation [13] which only requires shift and add operations. An additional clock cycle is needed for activation function; therefore results for the recall phase can be obtained in $J+I+4$ cycles.

Since target platform lacks dedicated multiplier blocks, a high-speed parallel multiplication scheme is required for the MAC. To minimize the propagation delay, multiplication module is implemented using Booth-Wallace Tree multiplier (Fig.3), where partial products generated with Booth radix-4 recorder are added with 4:2 compressors [14]. A 4:2 compressor adds four partial products (PPs) (*p1,…,p4*) to generate two updated PPs (*sum*, *c*) concurrently. For $N \times N$ bit multiplication, propagation delay using 4:2 compressors is estimated to be $3\log_{4/2}(N/4)$. Final stage addition for the multiplication scheme is performed by carry-lookahead adder to take advantage of the Virtex-E CLB's dedicated fast lookahead logic.

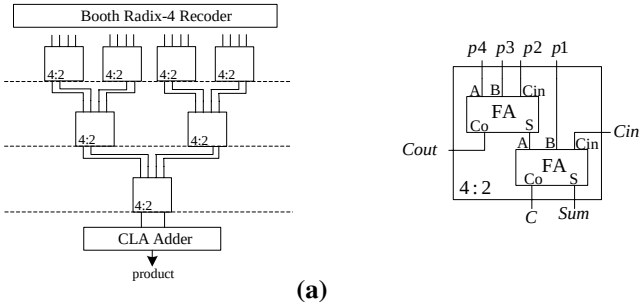

Fig. 3. **(a)** Booth-Wallace tree multiplier scheme with 4:2 compressors **(b)** The structure of 4:2 compressor

4 Synthesis Results

A VHDL model for the digital hardware part has been developed. Several recognition tasks have been tested on the design to verify the operation. Table 2 gives device utilizations for XCV600E illustrating the low area consumption of our design.

Table 2. Device Utilization for xcv600e-bg432

	Function Generators		CLB Slices		DFFs or Latches	
HW Blocks	Used	%	Used	%	Used	%
Pipelined MAC	4168	30.15	1941	28.08	1751	11.39
Activation Function	1235	8.93	711	10.28	54	0.35
SRAM	82	0.59	49	0.70	0	-
Total	5485	39.67	2701	39.06	1805	11.75

Performance evaluation for neurocomputers can be performed with two metrics; number of connections per second (CPS)[1] intended for recall phase, and number of connection updated per second (CUPS) intended for learning phase. For the 9:9:1 MLP architecture, the peak performance of our design for a single training pattern has been calculated and compared with other proposed implementations on Table 3 [15].

Table 3. Neural network implementations

Name	ANN TYPE	Neuron	Speed

[1] CPS=Connections Calculated/(Number of clock cycles required × Cycle Time)

RRANN	MLP	N/A	722 KCUPS
ECX	MLP/RBF	N/A	3.5 MCUPS
GRD (DSP)	Programmable	15	7 MCUPS
HNC 100-NAP	Programmable	100 PU	64 MCUPS
Hitachi WSI	Hopfield	576	138 MCPS
Our Design	**MLP**	**9 PU**	**335 MCPS**
Siemens MA-16	N/A	16 PU	400 MCPS
Innova	Programmable	64 PU	870 MCPS
AT&T Anna	MLP	16-256 PU	2.1 GCPS

5 Conclusions

This paper presents HW/SW codesign solution to eliminate the FPGA design bottleneck for feed-forward network architectures. The motivation for this study stems form the fact that an FPGA coprocessor with limited logic density and capabilities, Xilinx XCV600E, is able to act as a standalone device for pattern recognition tasks once the software partition handles the learning stage properly. Synthesis results indicate high speed operation with limited number of PUs which may be a significant contribution in designs where high throughputs can be obtained in low cost FPGAs.

References

1. Eldredge, J.G., Hutchings, B.L.: RRANN: The run-time reconfiguration artificial neural network. IEEE Custom Integrated Circuits Conference. (1994) 77-80
2. Ferrucci, A., Martin, M., Geocaris, T., Schlag M., Chan, P.K.: A Field-Programmable Gate Array Implementation of a Self-Adapting and Scalable Connectionist Network. FPGA'94: International ACM/SIGDA Workshop on Field-Programmable Gate Arrays. (1994)
3. Haenni, J.O., Beuchat, J.L., Sanchez, E.: Hardware reconfigurable neural networks, 5th Reconfigurable Architectures Workshop (1998)
4. Hikawa, H.: Implementation of simplified multilayer neural network with on-chip learning. Proceedings of the IEEE International Conference on Neural Networks. (1999) 1633-1637
5. Wolf, W.H.: Hardware-software co-design of embedded systems, Proceedings of the IEEE, Volume 82, Issue 7. (1994) 967-989
6. Seljak, B.K.: Hardware-software co-design for a real-time executive. Proceedings of the IEEE International Symposium on Industrial Electronics. (1999) 55-58
7. Pasero, E., Perri, M.: HW-SW codesign of a flexible neural controller through a FPGA-based neural network programmed in VHDL. IEEE International Joint Conference on Neural Networks, Volume 4. (2004)
8. Chiaberge, M., Miranda, E., Reyneri, L.M.: An HW/SW co-design approach for neuro-fuzzy hardware design. Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems. (1999)
9. Reyneri, L. M., Chiaberge, M., Lavagno, L., Pino, B., Miranda, E.: Simulink-based HW/SW codesign of embedded neuro-fuzzy systems. Int. J. Neural Syst., vol. 10, no. 3. (2000) 211-226

10. Xilinx Inc., Virtex-E 1.8 V Field Programmable Gate Arrays DS022-2 (v2.6.1) Production Product Specification. (2004)
11. Holt, J.L., Baker, T.E.: Backpropagation simulations using limited precision calculations. International Joint Conference on Neural Networks (IJCNN-91), vol. 2. (1991) 121-126
12. Kung, S.Y., Hwang, J.N.: Parallel Architectures for Artificial neural Nets. Proc. IEEE lnt. Conf. on Neural Networks, San Diego, California, (1988) pp. II-165 - H-172
13. Amin, H., Curtis, K.M., and Hayes-Gill, B.R.: Piecewise linear approximation applied to nonlinear function of a neural network. IEE Proceedings, Circuits Devices & Systems, 144 No. 6, December. (1997) pp. 313-317
14. Mori, J., Nagamatsu, M., Hirano, M., Tanaka, S., Noda, M., Toyoshima, Y., Hashimoto, K.: A 10 ns 54×54 b parallel structured full array multiplier with 0.5 m CMOS technology. IEEE Journal of Solid-State Circuits, Volume 26, Issue 4. (1991) 600-606
15. Lindsey, C., Lindblad, T.: Review of Hardware Neural Networks: A User's Perspective. Proceedings of 3rd Workshop on Neural Networks: From Biology to High Energy Physics. (1994)

Acknowledgement

The authors would like to thank the British Council for their support in this project. This study is a part of a British Council Partnership Programme funded project "The Implementation of Fuzzy Neural Networks for Phoneme Classification on Reconfigurable Gate Arrays".