International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 6, No3 (May-2017)
http://dx.doi.org/10.12785/ijcds/060305

Evaluation of Artificial Neural Network Architectures for Pattern Recognition on FPGA

Thang Viet Huynh

Electronic and Telecommunication Department, Danang University of Science and Technology – The University of Danang, Danang City, Vietnam

Received 28 Feb. 2017, Revised 19 Mar. 2017, Accepted 20 Apr. 2017, Published 1 May 2017

Abstract: In this paper, we present the design and implementation of two hardware architectures, namely MHL-ANN and SHL-ANN, for the realization of artificial neural networks on reconfigurable computing platforms like FPGA. We use 16-bit half-precision floating-point number format to represent the weights of the designed networks. The networks are synthesized and verified on Xilinx Virtex-5 XC5VLX-110T FPGA. We study the scalability and hardware resource utilization of the two proposed neural network architectures. For performance evaluation, the handwritten digit recognition application with MNIST database is performed, which reported a recognition rate of 90.88% when using an MHL-ANN architecture of size 20-12-10 and a recognition rate of 96.83% when using an SHL-ANN architecture of size 784-40-10. Experimental results showed that the SHL-ANN architecture is very potential for high performance embedded recognition applications.

Keywords: Artificial Neural Network; MNIST; FPGA; Floating-point; Pattern recognition.

1. INTRODUCTION

Artificial Neural Networks (ANNs) have attracted many applications and practical implementations in pattern recognition, machine learning, as well as in deep learning research areas in recent years [1-3]. ANN architectures demand a huge amount of parallel computing resources and memory, thereby requiring parallel computing devices like field programmable gate arrays (FPGAs). One of the main challenges for efficient implementations of ANNs on FPGA is the data representation of weights and activation functions. All integer, fixed-point and floating-point number formats can be used in hardware realization of ANNs, as shown in [4-7]. While fixed-point and integer representations can potentially bring improved execution performance in the forward computation, it is of great difficulty to train ANNs using those number formats in software to obtain the desired accuracy. It is shown that reduced-precision floating-point numbers is a suitable choice for the hardware implementation of neural networks on FPGA [6] and experiments showed that an optimal number format for single-FPGA implementation of ANNs is the 16-bit half-precision floating-point. Recent results of FPGA implementation of ANN for handwritten digit recognition have been reported in [7].

In this paper, we will further investigate various architectures for hardware implementations of ANNs and study the scalability of those architectures when they are synthesized and mapped on an FPGA device. The contributions of this paper are as follows:

- We present the hardware design of two 2-layer fully connected feed-forward ANN architectures, namely Multiple Hardware-Layer ANN and Single Hardware-Layer ANN, and shows the corresponding implementation results of these architectures for the handwritten digit recognition with MNIST database on the Xilinx Virtex-5 XC5VLX-110T FPGA.
- We investigate the scalability of the two proposed ANN architectures and make comparison between them in terms of hardware resource, peak performance and recognition accuracy.

The paper is organized as follows. Section 2 briefly presents the background. Section 3 details the design of two ANN architectures. Section 4 presents experimental results on practical FPGA device and makes performance comparison. Finally, we conclude the paper and introduce future research directions in Section 5

E-mail: thanghv@dut.udn.vn

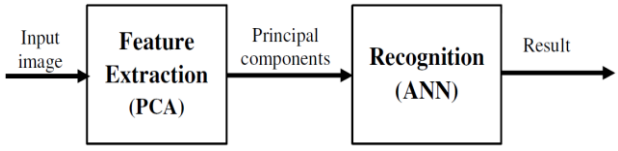

Figure 1. Recognition system based on ANN

2. BACKGROUND AND EXPERIMENT SETUP

Figure 1 presents block diagram of a typical recognition system, which consists of two functional units: *1)* a feature extraction unit using principal component analysis (PCA) technique and *2)* an ANN for recognition. For performance efficiency, these two units are expected to be located on the FPGA hardware. However, due to limited computing and storage resources of FPGAs, the PCA unit is normally implemented on microprocessor unit (as in [7]), leaving only the ANN implemented on FPGAs. For deep learning applications, the feature extraction unit is often removed and the inputs are used as features to the recognition unit using ANN.

TABLE I. MNIST RECOGNITION RATE (%) VERSUS ANN SIZE [6]

	Number of hidden neurons			
	12	16	20	24
Recognition rate (%)	91.5	92.1	93.5	93.9

In this work, we use the MNIST database [8] as the data set for training and testing the neural network. The MNIST database has a training set of 60,000 samples, and a test set of 10,000 samples of 28x28 (784 image pixels) gray-level images. There are 10 different handwritten digits ranging from 0 to 9 in the database. We perform recognition in two cases: with PCA as feature extractor and without PCA (i.e., using 784 image pixels as inputs to ANN). For recognition task, we employ 2 layer ANNs with optimal weights obtained from backpropagation training off-line on a desktop PC. The designed neural network has 10 neurons in the output layer, corresponding to 10 types of output digits. Table I (extracted from [6]) reports the recognition accuracy of proposed ANNs with 20 principal components as input features under varying number of hidden neurons. We choose to use a 20-12-10 ANN configuration (20 principal components as inputs, 12 hidden neurons, 10 output neurons) for the hardware implementation on Xilinx Virtex-5 FPGA.

Neural networks operate in two phases: *learning phase* that normally uses back propagation algorithm for obtaining the optimal weights, and *execution phase* that performs forward computation. In this work, we assume that the learning phase has completely been carried out offline on computers and only focus on the hardware realization of the forward computation of neural networks. The forward computation at each neural layer is described as follows [7]. Given the weight matrix $\underline{W}$ of size MxN of each neural layer and the input vector $\underline{x} = [x_1, x_2, ..., x_N]$ of length N at each neural layer, the output r_i of the *i-th* neuron is then computed by equations (1) and (2):

$$t_i = \sum_{j=1}^{N} w_{i,j} . x_j \tag{1}$$

$$r_i = f(t_i) = \frac{1}{1 + e^{-t_i}} \tag{2}$$

where: t_i is the net weighted sum of the *i-th* neuron; $f(t_i)$ is the *log-sigmoid* activation function (transfer function) of the neuron. Expanding equation (1) for all neurons in the layer, we can obtain the computation for the net weighted sum for one layer in a matrix-vector form as shown by equation (3).

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,j} & \dots & w_{1,N} \\ w_{2,1} & w_{2,2} & \dots & w_{2,j} & \dots & w_{2,N} \\ \vdots & & & \vdots & & \vdots \\ w_{i,1} & w_{i,2} & \dots & w_{i,j} & \dots & w_{i,N} \\ \vdots & & & \vdots & & \vdots \\ w_{M,1} & w_{M,2} & \dots & w_{M,j} & \dots & w_{M,N} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_i \\ \vdots \\ t_M \end{bmatrix} \tag{3}$$

The net weighted sum t_i of the *i-th* neuron is the dot-product between the input vector $\underline{x}$ and the *i-th row-vector* $\underline{w_i} = [w_{i,1}, w_{i,2}, \dots, w_{i,N}]$ of the weight matrix $\underline{W}$. The output column vector $\underline{r} = [r_1, r_2, ..., r_M]^T$ of the entire layer is calculated by applying the log-sigmoid activation function in a component-wise manner to the net weighted sum vector $\underline{t} = [t_1, t_2, ..., t_M]^T$.

For representing the weights of ANNs, we use the half-precision floating-point number format. The IEEE-754 standard [9] features the half-precision floating-point format with 1 sign bit, 5 exponent bits and 10 fraction bits, or 16 bits in total, which allows for saving of 2X and 4X the registers (memory) for weight storage compared to the single-precision (32 bits) and double-precision (64) ones. For FPGA implementation of dot-product and activation function of ANNs, we employ FloPoCo library [10] for VHDL code of half-precision floating-point operations.

3. DESIGN OF ANN ARCHITECTURES ON FPGA

In this section, we present the design and implementation of 2-layer feedforward neural networks for the handwritten digit recognition application with network configuration and optimal weights mentioned in previous section. Note, that forward computation in neural networks is data dependent: the output at hidden layer will be the input at output layer. Therefore, forward computation is performed in a sequential manner, leading to two possible hardware implementation approaches: *i)* implementing all layers on hardware for improved

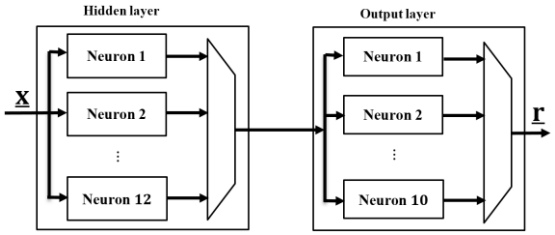

Figure 2. Functional block diagram of MHL-ANN architecture

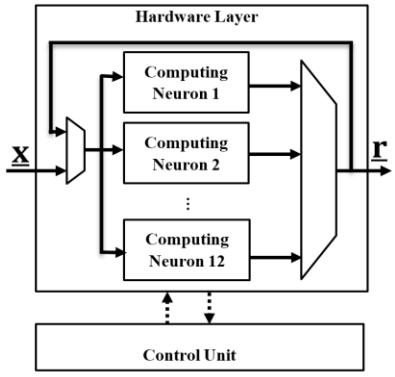

Figure 3. Functional block diagram of SHL-ANN architecture

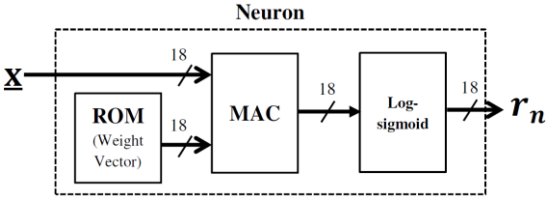

Figure 4. Architecture of one neuron

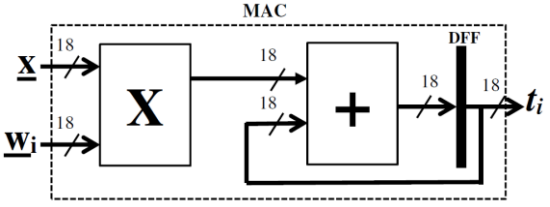

Figure 5. The Multiply-Accumulate (MAC) operation

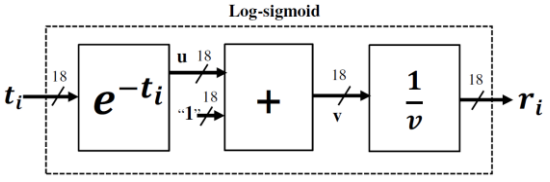

Figure 6. The Log-sigmoid operation

performance, or *ii)* implementing single layer on hardware with shared computing hardware circuitry for better area utilization. In this paper, we propose two architectures for FPGA hardware implementations of ANNs, as follows:

- MHL-ANN (Multiple Hardware-Layer ANN): all neurons in all layers (hidden layer and output layer) are implemented on hardware (Figure 2);
- SHL-ANN (Single Hardware-Layer ANN): one single hardware layer containing largest number of neurons is implemented on hardware (Figure 3); this hardware layer is used to perform the forward computations for both hidden layer and output layer.

A. *Multiple Hardware-Layer ANN*

The functional block diagram of the Multiple Hardware-Layer ANN (MHL-ANN) architecture is presented in Figure 2. The MHL-ANN consists of two layers with the same generic architecture. There are 12 neurons and 10 neurons in the hidden layer and output layer, respectively. The architecture of each neuron is detailed in Figures 4, 5, and 6 (adopted from [7]).

As shown in Figure 4, forward computation in each neuron is performed by the MAC operation followed by the Log-sigmoid operation. Figure 5 shows the functional block diagram of the MAC (Multiple-Accumulate) unit, which consists of a multiplier, an addition and a DFF (D flip-flop). The MAC unit performs the dot-product between two vector - the input vector $\underline{x}$ and the weight vector $\underline{w_i}$ – in a sequential manner, as presented by equation (1). The weight vector $\underline{w_i}$ of the neuron is stored in ROM. For saving FPGA resource, we reserve only 32 18-bit positions in the ROM in the current design of the MHL-ANN architecture, allowing for the execution of half precision floating point dot product of weight vectors with maximum length of 32 elements.

For the implementation of the activation function, we combine three operations – an exponential, an addition and an inverse - to perform the Log-sigmoid function as shown by equation (2). The hardware structure for the Log-sigmoid operation is specified in Figure 6. The implementation of half-precision operators is based on the FloPoCo library [10], which requires 2 more bits for the exception field; and the data-path will be 18-bit wires in VHDL implementation of the neural networks.

B. *Single Hardware-Layer ANN*

Figure 3 presents the functional block diagram of the Single Hardware-Layer ANN (SHL-ANN) architecture used to perform forward computation of a two-layer 20-12-10 ANN configuration. In contrast to MHL-ANN architecture, there is only one single *physical hardware layer* in the SHL-ANN architecture that consists of 12 *computing neurons* to perform the forward computation for both hidden and output neurons in the desired ANN.

The functional operation of each computing neuron in the SHL-ANN architecture is as follows. Computing neuron 1 is responsible for the computation of neuron 1 in the *logical hidden layer* and neuron 1 in the *logical output layer*, which will be executed sequentially. Other computing neurons act the same as computing neuron 1. The output of hidden layer is fed back as an input to the hardware layer to perform the forward computation for output layer, as shown in Figure 3. Since there are only 10 neurons in the logical output layer, two computing neurons 11 and 12 only perform the forward computation for neurons 11 and 12 in the logical hidden layer.

The hardware architecture of each computing neuron in SHL-ANN architecture is the same as the hardware architecture of the one implemented in MHL-ANN architecture, except that each computing neuron in SHL-ANN architecture will have to store two weight vectors for two appropriate neurons in hidden layer and output layer, respectively. The control unit is used to control the forward computation of logic neural layers, i.e., selecting the right weight vector and appropriate input vector to the physical hardware layer.

Since we intend to perform handwritten digit recognition with MNIST database consisting of 784-pixel images, we may increase the number of inputs to neural network up to more than 784. This allows for using those input directly, thereby removing the feature extraction with PCA module (see Figure 1). For doing so, we reserve up to 1024 (2^{10}) 18-bit positions in weight ROMs in SHL-ANN architecture, allowing for input vector length to neural network up to 1024 elements.

4. EVALUATION

We perform handwritten digit recognition with the MNIST database for performance evaluation. The neural network training is performed off-line using MATLAB on a desktop PC having 8 GB RAM and running at a system clock of 3.2 GHz under Windows 7. The optimal weight matrices obtained from the training phase are then used for code generation of VHDL specifications for hardware implementation on FPGA. For hardware synthesis and implementation, we use the Xilinx ISE tool 14.1 and Virtex-5 XC5VLX-110T FPGA board.

A. MHL-ANN versus SHL-ANN in MNIST

We compare the MHL-ANN architecture with the SHL-ANN architecture at the same neural network configuration 20-12-10 with respect to FPGA resources, execution time and recognition rate, as shown in Table II. The SHL-ANN implementation uses about 1.8x less registers, LUTs, BRAMs and DSP units than the MHL-ANN counterpart while guaranteeing the same recognition accuracy (90.88%) as well as (almost) the same execution time per image (509 cycles and 507 cycles).

Both implementations shown in Table II use only 20 principal features extracted from 784 original image pixels, meaning that a feature extraction unit with PCA technique is required. Note, that with respect to peak execution performance, the MHL-ANN implementation would potentially outperform the SHL-ANN implementation as the forward computations of hidden layer and output layer in the MHL-ANN architecture can be pipelined.

TABLE II. COMPARISON BETWEEN IMPLEMENTATIONS OF MHL-ANN AND SHL-ANN FOR MNIST DATABASE AT NETWORK CONFIGURATION 20-12-10 ON XILINX XC5VLX110-T FPGA (SPEED GRADE -3)

	MHL-ANN 20-12-10	SHL-ANN 20-12-10	Available
FFs	24025	13111	69120
LUTs	28340	15938	69120
BRAMs	22	12	148
DSPs	22	12	64
f_{max} (MHz)	205	188	-
Exec. time (cycles)	507	509	-
Recog. rate (%)	90.88	90.88	-

B. Scalability of MHL-ANN architecture

Since hardware implementation of ANN is expensive in terms of resources, determining the biggest neural network implementable on a given FPGA device is of importance. We investigate how well the two proposed ANN architectures scale with increased network size on the chosen Virtex-5 XC5VLX-110T FPGA. To do so, we vary the network size, generate VHDL models and implement the designs on FPGA using Xilinx ISE tool. We run both synthesis and implementation processes in Xilinx ISE software to verify if the designed network can really be implemented on the chosen FPGA.

TABLE III. IMPLEMENTATION RESULTS OF VARIOUS MHL-ANN CONFIGURATIONS ON XILINX VIRTEX-5 XC5VLX-110T; SPEED GRADE -3; TOTAL SLICE: 69,120.

MHL-ANN configuration	**Total number of neurons**	**Resource utilization (%)**	**Max. frequency f_{max} (MHz)**	**Implementable**
20-12-10	22	41	205	Yes
20-16-10	26	48	192	Yes
20-20-10	30	56	192	Yes
20-24-10	34	62	188	Yes
30-30-10	40	74	188	Yes
30-30-15	45	82	180	Yes
30-30-20	*50*	*92*	*180*	*Failed*

Table III reports the implementation results for MHL-ANN architecture with seven network configurations considered. Among them, the first four implementations follow the configurations for the handwritten digit recognition application reported in Table I. The last three network configurations aim to investigate whether the designed neural network is implementable on the chosen FPGA with respect to routing resources required for the connections in the whole network. Note, that the routing resource of a design on FPGA is normally not reported explicitly like the computing resource, therefore, it may happen that a design can be synthesized but cannot be implemented (*Place-and-Route*) on an FPGA due to limited routing resource.

Generally, Table III shows that the area cost for the MHL-ANN implementations scales proportionally with the network size (or total number of neurons). The average area cost for one neuron is about 1.85% of the total slices (69,120), or about 1300 slices per neuron. The maximum operating frequency of neural network decreases with increased network size, as expected. While the first six networks are successfully synthesized and implemented, the last one (configuration 30-30-20 with 50 neurons) can be synthesized but is failed to implement on the chosen FPGA device.

From our experiments, it is shown that an MHL-ANN with 45 neurons (30 inputs, 30 hidden neurons, and 15 output neurons) is currently the biggest neural network implementable on Virtex-5 XC5VLX-110T FPGA. Obviously, larger FPGAs like the Virtex-6 and Virtex-7 devices are needed for implementation of more complicated MHL-ANNs.

C. Scalability of SHL-ANN architecture

We investigate the scalability of the SHL-ANN architecture on the chosen Virtex-5 FPGA. The experimental results are shown in Table IV, for which all investigated neural network configurations are both synthesizable and implementable. Since the proposed SHL-ANN architecture can accept large input vector to the network, this allows for: *a)* removing the feature extraction unit in the recognition system shown in Figure 1, and *b)* using 784 image pixels as direct inputs. As shown in Table IV, increasing number of inputs to 784 pixels while still employing 12 hidden neurons can improve the recognition rate by 2.45% (from 90.88% to 93.33%) while it costs about 5% more hardware resource (from 23% to 28%) compared to the case when using 20 principal components as inputs.

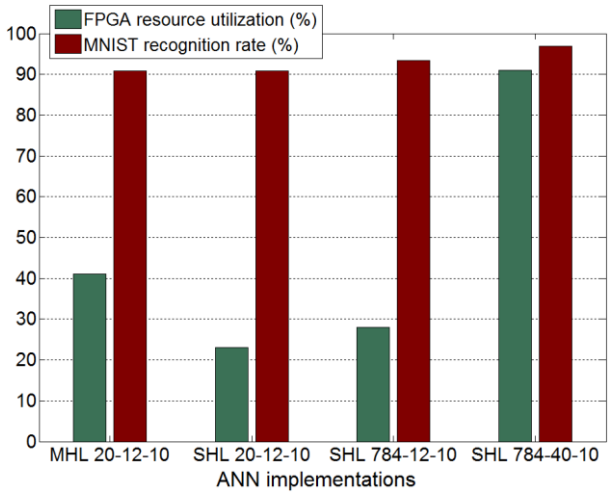

Figure 7. Comparison among various neural network implementations in terms of FPGA resource utilization (%) and recognition rate (%) (MHL: MHL-ANN, SHL: SHL-ANN)

TABLE IV. IMPLEMENTATION RESULTS OF SHL-ANN CONFIGURATIONS ON XILINX VIRTEX-5 XC5VLX-110T; SPEED GRADE -3; TOTAL SLICE: 69,120.

SHL-ANN configuration	Total number of neurons	Resource utilization (%)	Max. frequency f_{max} (MHz)	Recognition rate (%)
20-12-10	22	23	188	90.88
784-12-10	22	28	193	93.33
784-40-10	**50**	**91**	**180**	**96.83**

More importantly, it is shown in our experiments that an SHL-ANN implementation (configuration 784-40-10) having maximal of 50 neurons can be fitted on the Virtex-5 XC5VLX-110T FPGA chip and it offers the best recognition rate of 96.83%, i.e., only 317 misclassified images out of 10.000 test images in the MNIST database.

Figure 7 compares various neural network implementations with respect to FPGA hardware resource utilization (%) and corresponding MNIST recognition rate (%), which obviously shows that the SHL-ANN architecture is much more efficient than the MHL-ANN one.

5. CONCLUSION

We have presented the design, implementation and verification on FPGA of 2-layer feed-forward artificial neural network architectures for handwritten digit recognition system. We study the scalability and hardware resource utilization of our two proposed neural network architectures (MHL-ANN and SHL-ANN) on the Xilinx Virtex-5 XC5VLX-110T FPGA. Experimental results

showed that the SHL-ANN architecture is very potential for high performance embedded recognition applications. Future work will focus on improving performance of the proposed neural network architectures as well as extending the networks to other recognition applications like face recognition and/or image recognition.

ACKNOWLEDGMENT

The author would like to thank colleagues from ETE Department in DUT-UDN for their valuable supports. This work is carried out within the framework of the Ministry-level Scientific Research Project No. B2016-DNA-39-TT.

REFERENCES

[1] Janardan Misra, Indranil Saha, Artificial neural networks in hardware: A survey of two decades of progress, Neurocomputing, Volume 74, Issues 1–3, December 2010, pp 239-255.

[2] IBM Research: Neurosynaptic Chips: http://research.ibm.com/cognitive-computing/neurosynaptic-chips.shtml#fbid=U7U04FvCvJW (accessed 28.02.2017)

[3] C. Shi et al., "A 1000 fps Vision Chip Based on a Dynamically Reconfigurable Hybrid Architecture Comprising a PE Array Processor and Self-Organizing Map Neural Network," IEEE Journal of Solid-State Circuits, vol. 49, no. 9, pp. 2067-2082, 2014

[4] K. R. Nichols, M. A. Moussa, and S. M. Areibi, "Feasibility of Floating-Point arithmetic in FPGA based artificial neural networks," in *In CAINE*, 2002, pp. 8-13

[5] M. Hoffman, P. Bauer, B. Hemrnelman, and A. Hasan, "Hardware synthesis of artificial neural networks using field programmable gate arrays and fixed-point numbers," in *Region 5 Conference, 2006 IEEE*.

[6] T. V. Huynh, "Design space exploration for a single-FPGA handwritten digit recognition system," in IEEE ICCE, 2014

[7] T. V. Huynh, "Design of Artificial Neural Network Architecture for Handwritten Digit Recognition on FPGA", Journal of Science and Techonlogy, UDN, Vol. 11(108).2016. pp. 206-210, 2016

[8] MNIST database: http://yann.lecun.com/exdb/mnist/ (accessed 28.02.2016)

[9] "IEEE standard for floating-point arithmetic," IEEE Std 754-2008

[10] FloPoCo code generator: http://flopoco.gforge.inria.fr/ (accessed 28.02.2016)

Thang Viet Huynh received his PhD degree in Electrical and Electronic Engineering from Graz University of Technology (TU Graz), Austria in 2012. He is currently working as a lecturer at Danang University of Science and Technology (DUT), The University of Danang (UDN), Vietnam. His research interests include reconfigurable computing (FPGA), machine learning, NoC, and respective applications.