Deep Neural Network Accelerator based on FPGA

Thang Viet Huynh
Danang University of Science and Technology, The University of Danang, Vietnam
Email: thanghv@dut.udn.vn

Abstract — **In this work, we propose an efficient architecture for the hardware realization of deep neural networks on reconfigurable computing platforms like FPGA. The proposed neural network architecture employs only one single physical computing layer to perform the whole computational fabric of fully-connected feedforward deep neural networks with customizable number of layers, number of neurons per layer and number of inputs. The inputs, weights and outputs of the network are represented in 16-bit half-precision floating-point number format. The network weights are hard-coded using on-chip memory of FPGA devices, allowing for very fast computation. For performance evaluation, the handwritten digit recognition application with MNIST database is performed, which reported a recognition rate of 97.20% and a peak performance of 15.81 kFPS when using a deep neural network of size 784-40-40-10 on the Xilinx Virtex-5 XC5VLX-110T device. When implementing a deep neural network of size 784-126-126-10 for MNIST database on the Xilinx ZynQ-7000 XC7Z045 device, the recognition rate is 98.16% and the peak performance is 15.90 kFPS.**

Keywords — machine learning; deep neural network; MNIST; FPGA; floating-point;

I. INTRODUCTION

Over the past few years, Deep Neural Networks (DNNs) have been attracted many research and implementations in a very wide range of tasks and applications. In some practical recognition applications, specific hardware implementation of DNN is necessary when a general-purpose computer is not suitable due to constraints in speed, chip cost and energy consumption. While DNN models can easily be implemented by software on general-purpose computers, hardware implementations of DNN on silicon still expose many challenges, such as implementing the multiplication between the input vector with the weight matrix at each layer, implementing the activation function, storing the network weights, and choosing the suitable number format for computation [1].

In a fully-connected deep neural network architecture, which will be the focus architecture in this work, each layer requires one distinct memory space for its own weight matrix for performance efficiency. As a deep neural network usually have one to six hidden layers, the memory demand for the weight matrices of the whole network grows rapidly as the size of the network increases, as well as the memory accesses. In addition, for high-speed applications, the computations within each layer should be carried out in parallel in the silicon circuitry. Those factors heavily affect the performance and power efficiency of DNNs. In that context, the field programmable gate array (FPGA) is a promising hardware platform for the implementation of DNN accelerator. Undoubtedly, FPGA implementation of artificial (deep) neural networks is currently a very active research topic [2]–[6].

For hardware implementation of DNNs, the chosen number format for weights and activation function is critical to the recognition rate and performance, for which several number formats can be considered, such as fixed-point [2], [3], integer/binary representations [4], or floating-point [5]. While fixed-point and integer/binary representations can bring improved execution performance in the forward computation of the networks, it is of great difficulty to train DNNs for recognition applications, and, afterward, map the optimal weights of the DNN onto hardware [2]–[4]. In addition, the downside to high-performance characteristic of binarized neural networks is a drop in accuracy [4], in comparison to floating point networks. For those reasons, the floating-point arithmetic is chosen for FPGA implementation of DNNs in our work because of the superior accuracy offered by floating-point number format – which was early shown in our previous work in [5], [6] – as well as the simpler network training process with back propagation learning algorithm in floating-point number in the current software tools.

In this paper, we propose an efficient architecture for hardware implementation of the feed-forward DNN accelerator on FPGAs. The proposed DNN hardware architecture uses only one *single physical computing layer* to perform the whole computational fabric of fully-connected neural networks. The topology of the network can easily be customized with arbitrary number of layers (with up to 8 computing layers, a maximal number of 7 hidden layers), arbitrary number of neurons per layer and number of inputs thanks to a VHDL core generator for DNN. The inputs, weights and outputs of the proposed network are represented in half-precision floating-point number format with 16-bit operands. The proposed DNN hardware architecture is modeled by VHDL and synthesizable for FPGA. It can generally be used by any neural network based applications. To the best of our knowledge, this work is among the first ones to propose an efficient and customizable deep neural network architecture using floating-point weights. For performance evaluation on FPGA, we employ the MNIST database for handwritten digit recognition application targeted on two Xilinx FPGA boards: Virtex-5 XC5VLX-110T and ZynQ-7000 7Z045.

The rest of this paper is organized as follows. Section II presents the basic algorithm and the proposed DNN hardware architecture synthesizable for FPGA. We present the experimental setup and performance evaluation for the MNIST recognition application in Section III. Finally, Section IV gives some concluding remarks.

II. DEEP NEURAL NETWORK ARCHITECTURE

A. Algorithm

In a general feedforward DNN with multiple hidden layers, each layer k has an input vector $\mathbf{x}_k$ and a weight matrix $\mathbf{W}_k$. For an DNN with M computing layers, the input vector is forwardly propagated to the next layer to produce an output vector $\mathbf{r}_k$, which then becomes the input vector $\mathbf{x}_{k+1}$ of the next layer $k+1$, by first multiplying the weight matrix and then applying the activation function, as follows:

$$\mathbf{x}_{k+1} = f(\mathbf{W}_k\mathbf{x}_k); \qquad k = 1, 2, 3..., M, \qquad (1)$$

where the commonly used activation function is logistic sigmoid function defined as:

$$f(t) = 1/(1+e^{-t}). \qquad (2)$$

Since forward propagation in neural networks is data-dependent, the whole computation can be executed in a sequential manner, and therefore the computing hardware circuitry can be reused among different layers, leading to better hardware area utilization. The control algorithm for a sequential forward propagation can be described by the pseudo code in **Algorithm 1** as follows:

Algorithm 1. DNN forward computation

```
$\mathbf{x}_1$ = input;
for $k$ in $1$ to $M$ loop
    $\mathbf{x}_{k+1} = f(\mathbf{W}_k\mathbf{x}_k)$
end loop;
output = $\mathbf{x}_{M+1}$;
```

B. General DNN architecture

In this design, we build only one single *physical computing layer* that performs the forward propagation plus a *control unit* for switching among different *logical layers* of the neural network. Figure 1 presents the general block diagram of the DNN. The DNN_Layer implements the physical computing layer for forward propagation while the DNN_Control implements the switching fabric following the control algorithm presented in Algorithm 1. In our design, the inputs, weights and outputs of the network are represented in 16-bit half-precision floating-point number format. We employ FloPoCo code generator [7] for VHDL implementation of synthesizable floating-point operations. Note, that the data-width of input and output signals in half-precision floating-point format is 18 bits with 2 more bits for exceptional field representation (00 for zero, 01 for normal numbers, 10 for infinities, and 11 for NaN); these 2-bit field can easily be added and removed when necessary.

C. DNN physical computing layer

The detailed architecture of DNN_Layer is showed in Figure 2. DNN_Layer consists of S neurons for forward computation, a 2-to-1 multiplexer for the selection (controlled by X_control signal) of input to the neurons and an S-to-1 multiplexer for the sequential selection (controlled by R_control signal) of outputs of the previous logical layer fed to the next logical layer. The selections of inputs and outputs

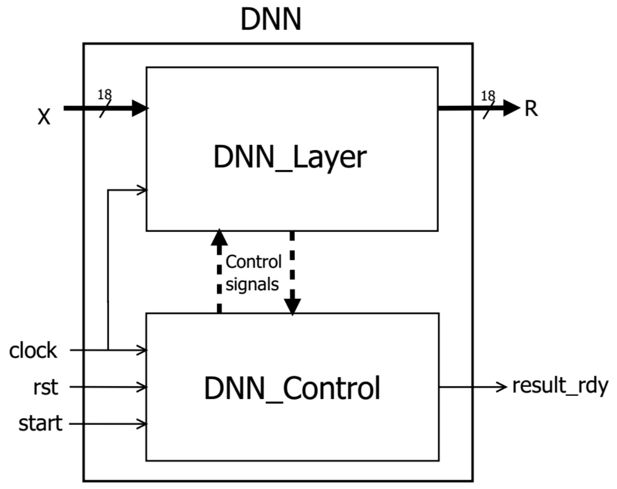

Fig. 1. General block diagram of DNN architecture

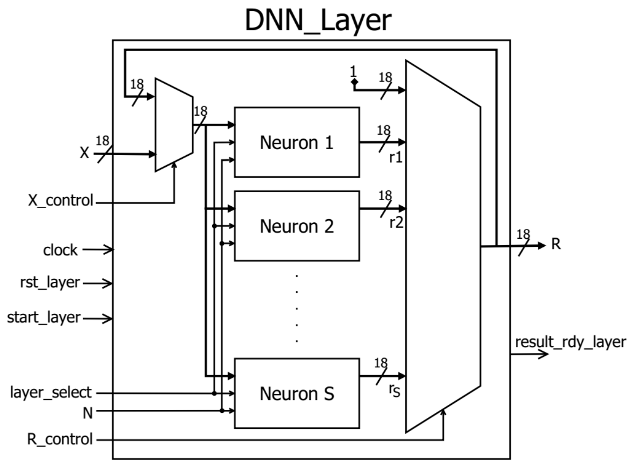

Fig. 2. Hardware architecture of DNN physical computing layer

are made by two control signals X_control and R_control. Other signals of the of DNN_Layer include rst_layer, start_layer, layer_select, N and result_rdy_layer; these signals are used to control the computation at each logical layer as well as data synchronization in the entire network.

The number of neurons implemented in DNN_Layer is the maximal number of neurons in the largest logical layer of the network. Each neuron in DNN_Layer consists of a weight memory that contains all the weight vectors corresponding to different logical layers. During computation of a neuron in DNN_Layer, the signal layer_select will select the right weight vector of that neuron corresponding to the right executing logical layer, while the signal N will define the number of inputs for that logical layer. When computing the multiplication of the weight matrix and the input vector at each layer, all hardware neurons in that layer execute in parallel and the input is fed to all neurons sequentially, i.e., the input is shared with all hardware neurons: the 1st input is fed to compute the 1st multiply-accumulate (MAC) for all the neurons, then the 2nd input is fed to compute the 2nd MAC for all the neurons, and so on.

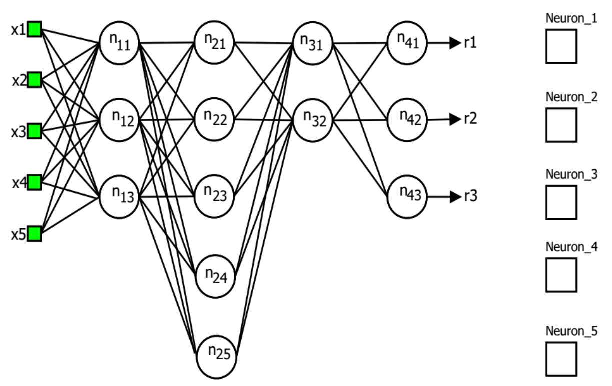

Fig. 3. Hardware architecture of DNN computing layer

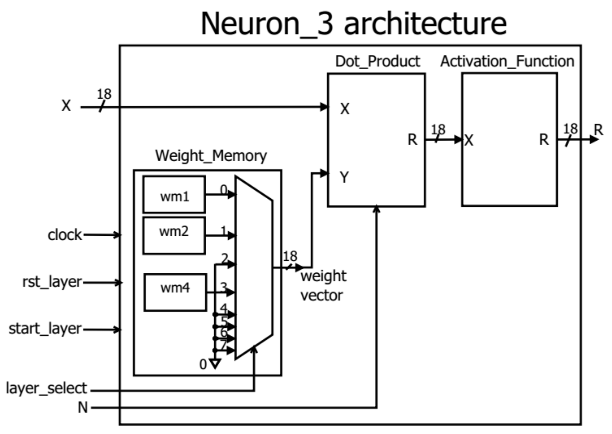

Fig. 4. Architecture of Neuron_3 in the DNN 5-3-5-2-3

D. Computing neuron

For better illustration, Figure 3 presents an example for a DNN of size 5-3-5-2-3 with 5 inputs, 3 outputs and 3 hidden layers. For this DNN configuration, there are 4 weight matrices for 4 logical layers (excluding input layer): $\mathbf{W}_1$ (3x6), $\mathbf{W}_2$ (5x4), $\mathbf{W}_3$ (2x6) and $\mathbf{W}_4$ (3x3). The maximal number of neuron is 5 (corresponding to the largest logic layer 2); therefore, the DNN_Layer will be implemented with 5 computing neurons. Each computing neuron will be responsible for the computations of appropriate neurons in all layers, for example, computing Neuron_1 performs the computations for logical neurons n_{11}, n_{21}, n_{31} and n_{41} in the logical layers 1, 2, 3 and 4, respectively. The timing synchronization for inputs and weight vectors for the accurate forward computation at each layer is guaranteed by layer_select and N signals controlled by DNN_Control unit.

Figure 4 details the architecture of the Neuron_3 corresponding to the DNN 5-3-5-2-3 implementation. As shown in Figure 4, the neuron contains a Dot_Product unit and an Activation_Function unit for the computation and a Weight_Memory unit for storing weight vectors. The Weight_Memory unit can have up to 8 sub-memory units for the weight vectors of up to 8 computing (logical) layers; these sub-memory units is selected by a an 8-to-1 multiplexer controlled by the layer_select signal. The Neuron_3 performs the computation for neurons n_{13}, n_{23} and n_{43} in layers 1, 2 and 4. There is no computation for Neuron_3 in the 3rd layer and it is, therefore, the Weight_Memory unit reserves only 3 sub-memory units (wm1, wm2 and wm4) for the appropriate weight vectors of layers 1, 2 and 4; the other inputs of multiplexer are connected to zero. Obviously, the DNN architecture efficiently uses on-chip memory for weight storage. This design strategy allows for best use of hardware area and BRAM of FPGA devices.

E. VHDL core generator for DNN

The proposed DNN hardware architecture is specified in VHDL. We then make MATLAB scripts for the auto-generation of DNN cores in VHDL synthesizable for FPGAs. The MATLAB-based DNN core generator allows users to quickly generate VHDL codes of their own deep neural networks (with up to 8 computing layers, with up to 7 hidden layers, arbitrary number of neurons per layer) in MATLAB environment given the weight matrices as inputs.

III. PERFORMANCE EVALUATION

A. Experimental setup

We carry out handwritten digit recognition problem with MNIST database [8] for performance evaluation of the DNN hardware architecture based on FPGA. The MNIST database has 60,000 samples for training and 10,000 samples for testing. It has 10 different handwritten digits from 0 to 9 in its database. Each digit is normalized and centered in a gray-level 28x28 image, resulting in a 784-input vector to the neural network. We use all 784 image pixels as direct inputs to the network. For DNN training, we conduct back propagation training with the Stochastic Gradient Descent (SGD) algorithm for multiple hidden layer neural networks. The SGD training is carried out off-line in MATLAB on a desktop PC. The optimal weight matrices obtained from the training are then used for the code generation of VHDL specifications for hardware implementation on FPGA. For hardware implementation, we use Xilinx ISE toolset 14.1 and employ Virtex-5 XC5VLX-110T and ZynQ-7000 7Z045 FPGA devices for synthesis and verification. The hardware resource of the two chosen FPGA devices is summarized in Table I.

B. FPGA resource usage

Based on the hardware resource required for one computing neuron (1092 FFs, 1311 LUTs, 1 BRAM and 1 DSP slice), we roughly estimate the maximal number of neurons implementable on the chosen FPGA devices. By synthesis and implementation in the ISE toolset, we observed that the maximal numbers of computing neurons fitted on the chosen FPGAs are 40 and 126 computing neurons for the Virtex-5 XC5VLX-110T and the ZynQ-7000 7Z045, respectively. From that observation, we vary the number of hidden layers to study the scalability and performance of the proposed DNN architecture. The detailed reports are shown in Table II, showing that the largest DNN implementations on the Xilinx devices are a 784-40-40-40-10 network for Virtex-5 chip and a 784-126-126-126-10 network for ZynQ-7000 chip, respectively. Both networks have 3 hidden layers and occupy

TABLE I. TOTAL HARDWARE RESOURCE OF XILINX VIRTEX-5 XC5VLX-110T AND ZYNQ-7000 7Z045 FPGA DEVICES

Device	FF	LUT	BRAM	DSP
Virtex-5 XC5VLX-110T	69120	69120	148	64
ZynQ-7000 7Z045	437200	218600	545	900

TABLE II. RESOURCE UTILIZATION OF DNN ARCHITECTURES ON VIRTEX-5 XC5VLX-110T AND ZYNQ-7000 7Z045 FPGA DEVICES.

DNN architecture	Computing neuron	FF	LUT	Resource utilization
784-40-10	40	44059	62579	90.6 (%)
784-40-40-10	40	44080	63454	91.8 (%)
784-40-40-40-10	40	44049	63591	92.0 (%)
784-126-10	126	139364	211692	96.8 (%)
784-126-126-10	126	139387	214583	98.1 (%)
784-126-126-126-10	126	139391	218528	99.9 (%)

TABLE III. PERFORMANCE EVALUATION AND MNIST ACCURACY ON VIRTEX-5 XC5VLX-110T AND ZYNQ-7000 7Z045 FPGA DEVICES.

DNN architecture	Execution time per frame (cycles)	f_{max} (MHz)	Peak performance (kFPS)	MNIST accuracy (%)
784-40-10	10805	180	16.67	97.07
784-40-40-10	**11371**	**179**	**15.81**	**97.20**
784-40-40-40-10	11937	178	14.98	96.76
784-126-10	11923	190	15.92	98.00
784-126-126-10	**13607**	**216**	**15.90**	**98.16**
784-126-126-126-10	15291	219	14.29	97.88

most of the hardware resource available (92.0% and 99.9%, respectively). From ISE synthesis reports, it is shown that the number of BRAMs and DSPs is equal to the number of computing neurons implemented. Interestingly, increasing number of hidden layers (from 1 to 3 hidden layers) only slightly increases the hardware resource usage, this is thanks to the share of computing neurons among different layers.

C. *MNIST accuracy and peak performance*

Table III reports the MNIST recognition accuracy, the execution clock cycle for one image frame, as well as the peak performance for handwritten digit recognition with MNIST. In general, the recognition accuracy would scale with the network size (number of neurons per layer) and network depth (number of hidden layers). It is true when increasing network size, but not with increasing network depth for MNIST database. Experimental results interestingly show that increasing number of layers does not necessarily help to substantially increase recognition rate for the MNIST database, as reported in Table III when using 3 hidden layers. Our experiments obviously suggest that using a DNN with 2 (two) hidden layers is the optimal design choice with respect to both recognition accuracy and peak performance for the MNIST problem. The corresponding accuracies are 97.20% when employing a 784-40-40-10 network on Virtex-5 XC5VLX-110T chip and 98.16% when employing a 784-126-126-10 network on ZynQ-7000 7Z045 chip. The peak performance are 15.81 and 15.90 thousand handwritten image frames per second (kFPS), respectively.

D. *Comparison to related work*

Compared to the neural network model in [2], the DNN architecture in this work allows for much more complicated recognition applications with much larger neural network sizes to be customized and implemented on FPGA. The MNIST recognition accuracies of 98.16% and 97.20% achieved in our work are much better than the MNIST accuracy of 95.8% reported in [4], and comparable with the best MNIST accuracy of 98.92% reported in [3]. The MNIST performance of our floating-point DNN is lower than those of fixed-point and binary-based neural networks in [3], [4]. However, with respect to complexity of neural network training, our DNN model allows for simpler training process, compared to other training processes in related work [2]-[4], with back propagation learning algorithm in floating-point number which can easily be performed by common tools.

IV. CONCLUDING REMARKS

In this work, we have proposed an efficient and customizable hardware architecture for implementation of feed-forward DNN accelerator on FPGA. The main idea is to use only one single physical computing layer to perform the computation of the whole neural network. Our proposed DNN architecture can provide adequate recognition performance with only relatively small network sizes, thereby bringing more performance and saving hardware resources and power. We have also developed a VHDL code generator for customizable and synthesizable DNN architectures on FPGA. This DNN core generator will publicly be available in near future for fast neural network hardware prototyping.

ACKNOWLEDGEMENT

The author would like to thank for valuable supports of colleagues from Electronic and Telecommunication Engineering Department, Danang University of Science and Technology – The University of Danang. This work is within the framework of the Ministry-Level Scientific Research Project No. B2016-DNA-39-TT.

REFERENCES

[1] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1–3, pp. 239–255, Dec. 2010.
[2] N. Nedjah, R. M. Silva, and L. de M. Mourelle, "Compact yet efficient hardware implementation of artificial neural networks with customized topology," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9191–9206, 2012.
[3] J. Park and W. Sung, "FPGA based implementation of deep neural networks using on-chip memory only," in *2016 IEEE ICASSP*, 2016, pp. 1011–1015.
[4] Y. Umuroglu *et al.*, "FINN," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, 2017, pp. 65–74.
[5] T. V. Huynh, "Design space exploration for a single-FPGA handwritten digit recognition system," in *2014 IEEE ICCE*, 2014, pp. 291–296.
[6] T. V. Huynh, "Evaluation of Artificial Neural Network Architectures for Pattern Recognition on FPGA," *Int. J. Comput. Digit. Syst.*, vol. 6, no. 3, 2017.
[7] "FloPoCo project." [Online] http://flopoco.gforge.inria.fr/.
[8] "MNIST Database." [Online] http://yann.lecun.com/exdb/mnist/.