FPGA BASED IMPLEMENTATION OF DEEP NEURAL NETWORKS USING ON-CHIP MEMORY ONLY

Jinhwan Park and Wonyong Sung

Department of Electrical and Computer Engineering
Seoul National University
Seoul 151-744 Korea
Email: jhpark@dsp.snu.ac.kr, wysung@snu.ac.kr

ABSTRACT

Deep neural networks (DNNs) demand a very large amount of computation and weight storage, and thus efficient implementation using special purpose hardware is highly desired. In this work, we have developed an FPGA based fixed-point DNN system using only on-chip memory not to access external DRAM. The execution time and energy consumption of the developed system is compared with a GPU based implementation. Since the capacity of memory in FPGA is limited, only 3-bit weights are used for this implementation, and training based fixed-point weight optimization is employed. The implementation using Xilinx XC7Z045 is tested for the MNIST handwritten digit recognition benchmark and a phoneme recognition task on TIMIT corpus. The obtained speed is about one quarter of a GPU based implementation and much better than that of a PC based one. The power consumption is less than 5 Watt at the full speed operation resulting in much higher efficiency compared to GPU based systems.

Index Terms— Deep Neural Networks, FPGA, fixed-point optimization

1. INTRODUCTION

Feed-forward deep neural networks (DNNs) show quite good performance in speech and pattern recognition applications [1, 2]. Real-time implementation of feed-forward deep neural networks demand a very large number of arithmetic and memory access operations, thus DNNs are usually implemented using GPUs (Graphics Processing Units) [3, 4]. GPU based implementations consume large power exceeding 100 Watt [5]. In addition, a GPU based system needs a PC that occupies a large space, which may not be suitable for embedded applications requiring small foot-print units.

There are several previous works on VLSI and FPGA based implementation of a DNN or a CNN (Convolutional Neural Network). The system in [6] stores the weights at the external DRAM, and can configure the algorithm fairly flexibly. However, this system demands a large number of external memory accesses, and the throughput is fairly limited. A full custom VLSI that employs thousands of processing units and stores the weights at the on-chip memory was developed in [7]. This custom VLSI based system can achieve a very high throughput and consumes small power, but is not flexible.

In a general feedforward deep neural network with multiple hidden layers, each layer k has a signal vector $\mathbf{y}_k$, which is propagated to the next layer by multiplying the weight matrix $\mathbf{W}_{k+1}$, adding biases $\mathbf{b}_{k+1}$, and applying the activation function $\phi_{k+1}(\cdot)$ as follows:

$$\mathbf{y}_{k+1} = \phi_{k+1}(\mathbf{W}_{k+1}\mathbf{y}_k + \mathbf{b}_{k+1}). \quad (1)$$

A deep neural network usually employs one to six hidden layers. One of the most general activation functions is the logistic sigmoid function defined as

$$\sigma(x) = \frac{1}{1+e^{-x}}. \quad (2)$$

In fully-connected feedforward deep neural networks, each weight matrix between two layers demands parameters whose capacity is determined by the product of units in the anterior and the posterior layers. Considering a DNN employing hidden layers with 1,000 units, each weight matrix demands about one million parameters. This means that a few million weights are needed for the implementation of a typical DNN and more than 10MB of memory is needed when the weights are represented in the floating-point format. The number of output signals and that of biases are both proportional to the layer size. The signal word-length affects the complexity of arithmetic units and interconnection networks.

Reducing the complexity of neural networks using quantization or pruning has been studied much [8–13]. Instead of direct weight quantization, retraining with backpropagation was developed in [7, 14, 15]. The designed networks employed usually 2-8 bits for the weights and represented the signals in analog or high precision fixed-points using more than 7 bits. Recently, a research work that tries to increase the sparseness of the weights by pruning out small valued ones has been developed in order to reduce the model size and the execution time with a CPU [12, 13].

In this paper, we have developed a DNN using an off-the-shelf Xilinx FPGA, XC7Z045 aiming for design flexibility, high throughput with thousands of processing units, and low-power consumption with virtually no DRAM accesses by storing all the weights on on-chip memory. Since the on-chip memory of an FPGA is the most limited resource for the implementation of a DNN, we employ the training based weight quantization scheme and achieve a quite good performance only with 3 bits for the weight representation [13]. A handwritten digit recognition and phoneme recognition problems are implemented in this FPGA. It is, however, takes less than a few hours to develop a DNN with a different network configuration.

This paper is organized as follows. In Section 2, we describe the algorithm and the architecture for this FPGA based DNN system. Section 3 describes the implementation detail of this work. The experimental results are shown in Section 4. Concluding remarks are shown in Section 5.

This work was supported in part by the Brain Korea 21 Plus Project and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A1A10056051).

2. ALGORITHM AND ARCHITECTURE OPTIMIZATION

2.1. Fixed-Point DNN design

The DNN for handwritten digit recognition has three hidden layers, and the layer configuration is 784-1022-1022-1022-10. The input of the DNN is a 28×28 (=784) pixel image represented in 8-bit gray scale. The output corresponds to the likelihood of each digit. This algorithm needs approximately 3 million weights, which is the sum of $784 \times 1022 + 1022 \times 1022 + 1022 \times 1022 + 1022 \times 10$.

The DNN for phoneme recognition has four hidden layers. Each of the hidden layers has 1022 units. The input layer of the network has 429 units to accept 11 frames of MFCC (Mel-frequency cepstral coefficient) parameters. The output is the likelihood of 61 phonemes.

The FPGA used for this implementation is Z-7045 of Xilinx. This FPGA contains 218K LUTs (Look Up Tables), 2.18 MB of block RAM (BRAM), and 900 digital signal processing (DSP) blocks in addition to dual A9 ARM Cortex CPUs. Even if we employ 8 bits for representing each weight, the BRAM cannot accommodate all the weights and thus many DRAM accesses are needed.

In order to store all the weights on the FPGA, we apply fixed-point optimization for the weights, and successfully reduce the word-length into 3 bits. The weights are trained at the off-line and down-loaded to this system. The network training consists of three steps; the first one is an ordinary floating-point training, the second is the optimal uniform quantization minimizing the error in L2 norm, and the final step is the retraining with fixed-point weights.

For digit classification, the network is pre-trained with unsupervised greedy RBM (Restricted Boltzmann Machine) learning. Each layer is pre-trained by 50 epochs of 1-step contrastive-divergence based stochastic gradient descent with the mini-batch size of 100, the learning rate of 0.1, and the momentum of 0.9. Then, we ran 100 epochs of the backpropagation with stochastic gradient descent using the mini-batch size of 100, the fixed learning rate of 0.1, and the momentum of 0.9. MNIST dataset is used as the training set.

For phoneme recognition, each layer is pre-trained by 50 epochs of stochastic gradient descent with the mini-batch size of 128, the learning rate of 0.05, and the momentum of 0.9. We ran 100 epochs of the backpropagation using the mini-batch size of 128, the fixed learning rate of 0.05, and the momentum of 0.9 for fine tuning. TIMIT database is used for training. Dropout is applied for training both of networks.

After obtaining the floating-point weights, we apply fixed-point optimization that employs retraining after quantization [14]. The same training parameters are used for retraining weights. The weights of fixed-point DNN employ 3 bits for the input and hidden layers, and 8 bits for the output layer that is more sensitive to quantization. By this fixed-point optimization, we can obtain the miss-classification rate (MCR) of 1.08% [14]. Note that the DNN with 1022 floating-point units show the MCR of 1.06%, and the DNN with 900 floating-point units yields the MCR of 1.12%. For the phoneme recognition, the phoneme error rate of the floating-point DNN is 27.81%, while that of the 3-bit fixed-point DNN is 28.39%.

2.2. Parallel-serial architecture

The DNN for digit recognition needs approximately 3K neuron-like processing units: 1022 units for implementing hidden layer1, 1022 for hidden layer2, 1022 for hidden layer3, and 10 for the output layer. Each neuron-like processing unit conducts approximately 1022 multiply and add operations to process one input image. As a result, the total amount of computation for processing one image is approximately 3 million operations.

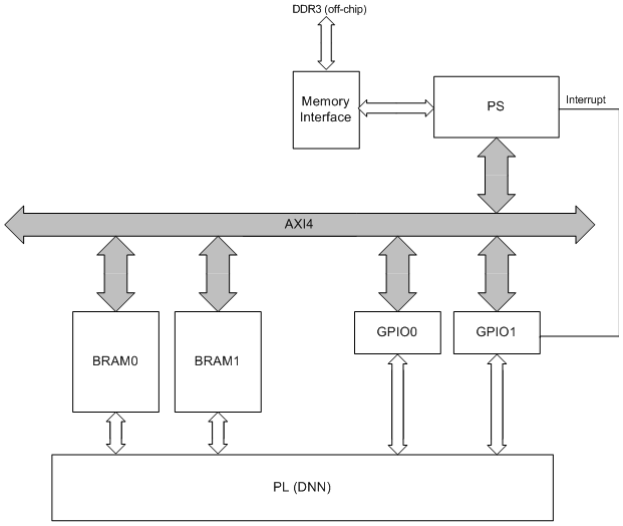

Fig. 1. System Overview.

In order to trade the throughput and the hardware resources, we employ the parallel-serial architecture. In this design, each layer contains 1022 processing units (PUs), which means parallel PU operations, and each PU computes the output in 1022 clock cycles, which is the serial processing in each PU. At each clock cycle, the PU conducts only one multiply-add operation. In fact, since the weights are quantized to 3 bits in the input and hidden layers, the PUs do not need multipliers. However, the synthesis shows that this design results in overuse of LUTs. As a result, the number of PUs is halved and the processing time becomes doubled as 2044 + some overhead cycles. With the system clock frequency of 100 MHz, this design can compute approximately 48.9 K outputs per second.

3. FPGA BASED IMPLEMENTATION

Figure 1 shows the overall block diagram of the system. The processing system (PS) and the programmable logic (PL) communicate through the general purpose IO (GPIO). For digit recognition, one hundred images are recognized at each operation in this design. However, the number of images to process at each batch can be changed easily. The operations are performed as follows.

- PS moves 100 handwritten images stored at the DRAM to BRAM0 (or BRAM1 alternatively), and activates the start signal in GPIO0.
- PL conducts the recognition of 100 images, writes the result to BRAM0 (or BRAM1 alternatively), and then activates the 'Done' signal in GPIO0 when the recognition process is finished. This process takes approximately 200K cycles.

As the operation is explained in the above, the PS and PL work in an interleaved manner using BRAM0 and BRAM1. The system for phoneme recognition operates in a similar way.

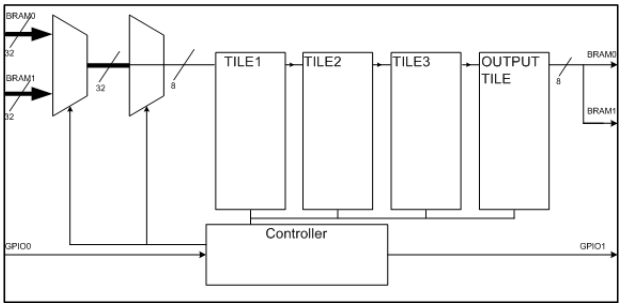

Fig. 2. Architecture of deep neural network circuit.

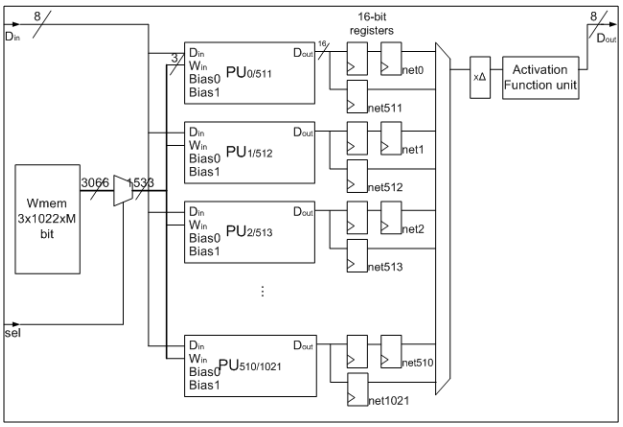

Fig. 3. Structure of a tile. M = 1022 for this figure.

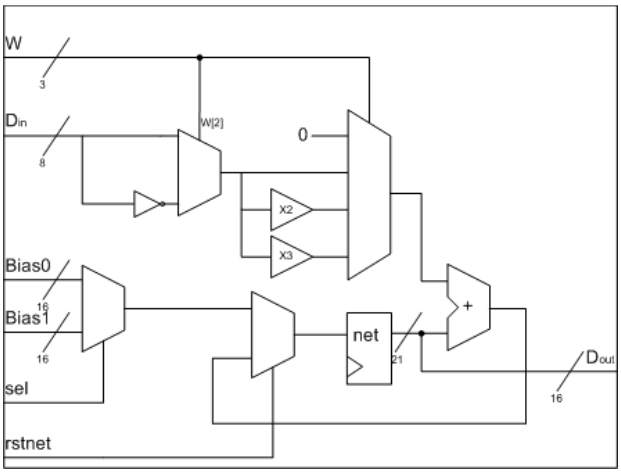

Fig. 4. Structure of a processing unit.

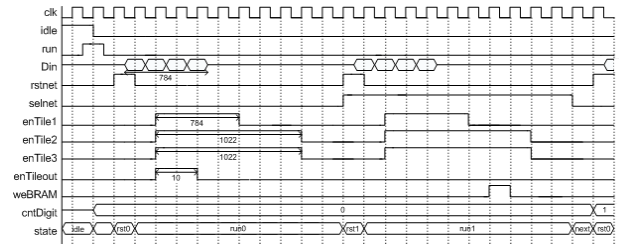

Fig. 5. Timing diagram of control signals.

Figure 2 shows the overall architecture of the circuit, where each tile implements a layer of the DNN algorithm. The input and output of the tile employ 8-bit word-length for signal representation. The structure of a tile is shown in Fig. 3. The tiles used for hidden layers have 511 (=1022/2) PUs each. Since each PU conducts the operation for two nodes, it has two output registers. Each tile contains BRAM for weight storage. Only one activation function is implemented in each tile.

The output tile contains a small number of output nodes, and employs 8-bit weight values. Since the number of nodes at the output tile is small, each PU conducts the operations that correspond to the function of one neuron. The result of the output node is compared to find the recognized digit or phoneme.

Figure 4 shows the structure of each PU. Each PU multiplies the input with the weight of -3 to 3, and adds the multiplied result to the partial sum stored in the accumulation register. According to the value of the weight, -3Din, -2Din, -Din, 0, Din, 2Din, or 3Din is added to the partial sum. This logic is implemented using LUTs without consuming any DSP block. The word-length of the adder in the PU is 16 bits. The final output of each PU is multiplied with the coefficient Δ, then the result (8-bit) is delivered to the next layer through the activation function.

The activation function is also implemented using combinational logic circuits. The minimized sum-of-product representation is used for the sigmoid function as proposed in [16]. We employ 8 bits for both input and output signals of the activation function.

The timing diagram of the operation is shown in Fig. 5. At first, 'rstnet' signal initializes the PUs. Since the value of 'selnet' is 0, each PU is initialized as its Bias0 value. After then, each tile receives 511 inputs from the previous tile and conducts the weight and accumulation operations using all the PUs in the tile. Note that it takes 2 clock cycles for processing one input because one PU implements two nodes. After then, the 'selnet' becomes 1, and the next 511 input is processed using Bias1 and the corresponding weights. After finishing this, the 'cntDigit' increases, and the next image is processed at the same way. When the cntDigit becomes 100, 'fin' signal is generated and the PL becomes in the idle state. In order to recognize one image, this circuit consumes 2063 (=2×1022 + overhead) clock cycles.

4. EXPERIMENTAL RESULTS

This work is designed using Xilinx Vivado (v.2014.4), and implemented on a Xilinx ZC706 evaluation board. This board contains XC7Z045 FPGA that contains both ARM CPUs and configurable logic circuits. The CPU frequency is 800 MHz. The Programmable Logic (PL) operates at 172 MHz for digit classification and 140 MHz for phoneme recognition. The FPGA resource utilization is shown in Table 1, where the designs with and without DSP slices are compared. Using DSP slices does not reduce the demand of FFs (flip-flops) and LUTs much. For comparison, resource utilization of the implementation with 8-bit fixed-point weights is given. It consumes almost all of LUTs and entire DSP slices on the FPGA chip because 8-bit weights need hardware multipliers for the implementation. The 8-bit representation for weights cannot be implemented on this FPGA because the BRAM capacity is not sufficient.

Table 1. FPGA resource utilization for digit recognition with different weight precision.

Resource	FF	LUT	BRAM	DSP
3-bit without DSP	130237	124862	323	0
3-bit with DSP	130802	121173	323	900
8-bit fixed point	136677	213593	750.5	900
Available	437200	218600	545	900

Table 2. FPGA resource utilization for phoneme recognition.

Resource	FF	LUT	BRAM	DSP
Used	161923	137300	378	0
Available	437200	218600	545	900

The time spent for recognizing 10,000 images is measured as 142 msec. This implementation can process about 70,000 images at each second. If weights were stored in DRAM, the memory bandwidth of 630 Gbit/sec (=3×3M×70000) is required to achieve this throughput. The memory bandwidth of the DRAM in Xilinx ZC706 board is 102.4 Gbit/sec. It is possible to reduce the number of DRAM accesses by processing multiple images at a time, which however demands extra internal registers for storing intermediate results and incurs more delay.

For the phoneme recognition application, the time spent for recognizing 10,000 frames is measured as 151 msec. This can process about 66,000 frames per second. Since a real-time speech recognition mostly uses 10 ms as for the frame size, this implementation result translates 660 times of speed-up over the real-time. One FPGA can conducts phoneme recognition needed for real-time speech recognition of over 600 people.

The power consumption shown at the simulation tool is shown in Table 3. The real power measurement of the evaluation board is 11.4 Watt and 13.1 Watt for digit and phoneme recognition, respectively, which includes all the power consumption in the peripherals and the power circuit. As shown in this table, the power consumed in the recognition circuit is about 5 Watt for obtaining the throughput of 70,000 images/sec. This translates 71 μJ for one image recognition.

We compare this implementation with that of a high-end GPU, the NVIDIA GeForce Titan Black. The GPU can recognize 250K images at each second, which is about 3.6 times higher than that of this FPGA based implementation. Note that the GPU used floating-point arithmetic for this test. But the accuracy difference is very minimal. However, the GPU system not only demands a much bigger physical space but also consumes much higher power. The power consumption of the GPU system is about 250 Watt [5]. Thus, the energy efficiency of this FPGA based system is considered to be over 10 times higher than that of the GPU based system.

During the last a few years, many DNN algorithms have been developed. Some of them is more complex than this design. A very complex DNN used for phoneme recognition demands 20 million weights [17]. Thus, this DNN circuit needs about 60 Mbits of BRAM to operate using 3-bit weights stored on on-chip memory. The FPGA density is growing very rapidly as well. Table 4 shows the hardware resources available in recent Xilinx FPGA families. Note that the FPGA employed to this design is equivalent to the Kintex-7 family. This table shows that even a very complex DNN algorithm can be implemented using an FPGA with only on-chip memory. More noticeable is the speed. With only 124K LUTs, the design achieved about 28% speed of a high-end GPU based system. If we employ the highest density FPGA containing 2533K LUTs, it would be possible to acheive more than 5 times of the speed-up compared to the GPU based system. The IO bandwidth bound is not a limitation because no DRAM access is needed.

Table 3. On-Chip power consumption (Watt) estimated by Xilinx Vivado.

Usage	Digit classification	Phoneme recognition
Clocks	0.579	0.528
Signals	1.202	3.559
Logic	0.849	2.810
BRAM	0.436	1.308
PS7	1.623	1.625
Device Static	0.291	0.357
Total	4.982	9.830

Table 4. Hardware resources in Xilinx FPGA families.

	Kintex-7	Virtex-7	Kintex UltraScale	Virtex UltraScale
LUTs (K)	299	1221	663	2533
Block RAM (Mb)	34	68	76	132.9
DSP slices	1920	3600	5520	2880

5. CONCLUDING REMARKS

We show an FPGA based implementation of a deep neural network for the handwritten digit recognition and the phoneme recognition problems, which needs millions of weights and arithmetic operations for producing one output. In order to use only on-chip memory for weight storage, the weights are represented in 3 bits, while the internal signals employ the precision of 8 bits. A retrain based fixed-point weight optimization technique is employed to reduce the performance gap with floating-point algorithms. The implemented FPGA shows the throughput that is about one quarter of a GPU based system, but only demands approximately 2~4% of the power consumption of the GPU system, resulting in over 10 times of power efficiency. If we employ a larger FPGA, such as Virtex-7 or Virtex UltraScale, a much higher throughput than that of a GPU based system can be achieved.

6. REFERENCES

[1] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[2] Geoffrey E Hinton and Ruslan R Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] Zhilu Chen, Jing Wang, Haibo He, and Xinming Huang, "A fast deep learning system using GPU," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1552–1555.

[4] Ying Zhang and Saizheng Zhang, "Optimized deep learning architectures with fast matrix operation kernels on parallel platform," in *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*. IEEE, 2013, pp. 71–78.

[5] NVIDIA, "GeForce GTX TITAN specifications," available: `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications`, [Online; accessed 25-Sep-2015].

[6] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 257–260.

[7] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung, "X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 7510–7514.

[8] Janardan Misra and Indranil Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1, pp. 239–255, 2010.

[9] Emile Fiesler, Amar Choudry, and H John Caulfield, "Weight discretization paradigm for optical neural networks," in *The Hague'90, 12-16 April*. International Society for Optics and Photonics, 1990, pp. 164–173.

[10] Perry Moerland and Emile Fiesler, "Neural network adaptations to hardware implementations," Tech. Rep., IDIAP, 1997.

[11] Chuan Zhang Tang and Hon Keung Kwan, "Multilayer feedforward neural networks with single powers-of-two weights," *Signal Processing, IEEE Transactions on*, vol. 41, no. 8, pp. 2724–2727, 1993.

[12] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[13] Dong Yu, Frank Seide, Gang Li, and Li Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4409–4412.

[14] Kyuyeon Hwang and Wonyong Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[15] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1131–1135.

[16] MT Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," in *Computers and Digital Techniques, IEE Proceedings-*. IET, 2003, vol. 150, pp. 403–411.

[17] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton, "Acoustic modeling using deep belief networks," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 14–22, 2012.