X1000 REAL-TIME PHONEME RECOGNITION VLSI USING FEED-FORWARD DEEP NEURAL NETWORKS

Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung

Department of Electrical and Computer Engineering, Seoul National University
Gwanak-gu, Seoul 151-744 Korea
E-mail: jhkim@dsp.snu.ac.kr, khwang@dsp.snu.ac.kr, wysung@snu.ac.kr

ABSTRACT

Deep neural networks show very good performance in phoneme and speech recognition applications when compared to previously used GMM (Gaussian Mixture Model)-based ones. However, efficient implementation of deep neural networks is difficult because the network size needs to be very large when high recognition accuracy is demanded. In this work, we develop a digital VLSI for phoneme recognition using deep neural networks and assess the design in terms of throughput, chip size, and power consumption. The developed VLSI employs a fixed-point optimization method that only uses +Δ, 0, and -Δ for representing each of the weight. The design employs 1,024 simple processing units in each layer, which however can be scaled easily according to the needed throughput, and the throughput of the architecture varies from 62.5 to 1,000 times of the real-time processing speed.

***Index Terms*—** Deep neural network, fixed-point optimization, phoneme recognition, VLSI

1. INTRODUCTION

Feed-forward deep neural networks (DNNs) employ multiple hidden layers, and they show quite good performance in speech and pattern recognition applications [1, 2]. Figure 1 shows a deep neural network composed of one input layer, four hidden layers, and one output layer.

The output of the $(k+1)$-th layer, $\mathbf{y}_{k+1}$, is computed using Eq. (1) and (2), where $\mathbf{y}_k$, $\mathbf{W}_{k+1}$, $\mathbf{b}_{k+1}$, and $\mathbf{net}_{k+1}$ are the input signal vector, weight matrix, bias vector, and net vector, respectively.

$$\mathbf{net}_{k+1} = \mathbf{W}_{k+1}\mathbf{y}_k + \mathbf{b}_{k+1} \tag{1}$$

$$\mathbf{y}_{k+1} = f_{k+1}(\mathbf{net}_{k+1}). \tag{2}$$

Note that $f_{k+1}(\cdot)$ represents the activation function, and the sigmoid function represented in Eq. (3) is usually used.

$$f_{k+1}(x) = \frac{1}{1+e^{-x}} \tag{3}$$

The operation of a DNN algorithm is quite memory access intensive because it employs all-to-all weighted connections between the units in adjacent layers. When the number of units in each layer is 1,024, the size of parameters for representing one hidden layer, $\mathbf{W}_k$, becomes 1 million. As a result, a practical DNN demands millions of weights, which are usually represented in the floating-point format. Most of the DNN applications are currently implemented using CPUs or GPUs [3, 4].

This work was supported in part by the Brain Korea 21 Project and the National Research Foundation of Korea (NRF) grants funded by the Ministry of Education, Science and Technology (MEST), Republic of Korea (No. 2012R1A2A2A06047297).

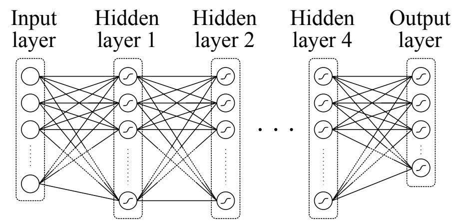

Fig. 1: A deep neural network with four hidden layers.

VLSI-based implementation of DNN algorithms is demanded for high-throughput and low-power applications. There are several, such as analog, digital, hybrid, and FPGA implementation approaches for neural networks [5, 6, 7, 8, 9, 10, 11]. However, the number of neurons in a layer is restricted to a small number in those implementations. We choose the digital VLSI-based architecture not only to exploit the convenience of circuit design but also to utilize high-density memory. Since the computing structure of the feed-forward DNN is very similar to the matrix-vector multiplication, the algorithm can be implemented using many processing elements. However, it is very needed to design a chip that contains all the weights in order to minimize external memory accesses.

In this study, we design a VLSI for DNN-based phoneme recognition. The algorithm needs approximately more than 5 million weights and 500 million arithmetic operations per second for real-time operation. In order to store all the weights on a small CMOS chip, the precision of the weights are reduced to only 3 levels (+Δ, 0, and -Δ) by employing the fixed-point optimization method described in Section 2. The developed architecture assigns one processing element to each unit of the DNN algorithm in order to maximize the parallel factor and increase the throughput. The developed VLSI can process one frame (10 millisecond) of speech in 0.01 millisecond using 1,024 cycles of 102.4 MHz clock, which translates about 1,000 times of the real-time processing speed.

This paper is organized as follows. In Section 2, a fixed-point deep neural network for phoneme recognition is developed. Section 3 describes the VLSI architecture of the developed fixed-point DNN. We show the synthesis and simulation results of the developed VLSI in Section 4. Finally, concluding remarks follow in Section 5.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

2. FIXED-POINT DNN DESIGN FOR PHONEME RECOGNITION

We use the DNN-based phoneme recognition algorithm that consists of an input layer, four hidden layers, and one output layer as illustrated in Fig. 1. The input layer contains 429 linear units to accept real-valued inputs that correspond to 11 frames of MFCC (Mel-frequency cepstral coefficient) parameters. Each of the four hidden layers contains 1,024 logistic units. The output layer consists of 61 logistic units that correspond to 61 target phoneme labels. The similar structure can be found in [4]. This design implements monophone recognition, but it can be easily extended to triphone based one by increasing the number of units in the output layer.

The network is pre-trained with unsupervised greedy restricted Boltzmann machine (RBM) learning [2]. The linear-logistic RBM is trained by 40 epochs under the learning rate of 0.005. For the other RBM, we use 20 epochs of 1-step contrastive-divergence-based stochastic gradient descent with the mini-batch size of 128, the learning rate of 0.05, and the momentum of 0.9. For the fine-tuning, we use 10 epochs of the back-propagation with the stochastic gradient descent, the mini-batch size of 128, the fixed learning rate of 0.05, and the momentum of 0.9.

For VLSI-based implementation of a DNN, fixed-point arithmetic is much desired [12]. However, direct quantization of the trained floating-point weights does not yield good results. Therefore, we employ the weight quantization strategy similar to the algorithms proposed in [7, 13] to retrain the weights after the direct quantization. Also, internal signals (output values of the units) are uniformly quantized from 0 to 1.

Initial fixed-point weights are obtained by directly quantizing the trained floating-point weights. The units in each layer share the same quantization step size Δ, which is determined using an optimization technique to minimize L_2 error of weights followed by exhaustive search of the parameter. In order to further refine the fixed-point weights, the back-propagation-based retraining algorithm is reapplied. In the retraining procedure, we maintain both the high- and low-precision weights and signals to accumulate the effects of small adaptation error. We only use two bits for representing 3-level weights (+Δ, 0, and -Δ) and four bits for internal signals.

Recently, a new training algorithm that intentionally drops out some of the connections in the network to prevent early over-fitting was developed [14]. In this algorithm, some processing elements that are randomly chosen are forced to have zero output. This algorithm shows excellent performance with floating-point arithmetic. We found that this algorithm also yields better recognition performance when applied to fixed-point DNN optimization.

Table 1 shows the performance of floating-point and fixed-point DNNs for phoneme recognition. The conventional training does not employ the drop-out during the back-propagation. For this training, we use the TIMIT corpus that is comprised of a training set from 462 speakers and a test set from 168 speakers [15]. All SA recordings, utterances of the same sentences from every speaker, in the corpus are removed during training since it can give bias to the results. The input receives 39 dimension MFCCs, which are 12th-order MFCCs with energy and their first and second temporal derivatives. MFCCs are extracted using the 25-ms Hamming window with the 10-ms frame rate. We use 11 consecutive frames, and the parameters are normalized to have zero mean and unit variance [4]. The evaluation uses 39 phones which are mapped from the original 61 phones as described in [16].

Table 1: Frame level phoneme recognition error rate according to the training method with floating-point and fixed-point arithmetic.

Hidden layer size and training method	Error rate (%)	
	Floating-point	Fixed-point
1,024-unit-layer with conv. training	26.24	27.63
1,024-unit-layer with drop-out training	23.71	24.93

3. HARDWARE ARCHITECTURE

The computing structure for each layer of a DNN is similar to the matrix-vector multiplication shown in Eq. (1), where the weights are stored in the matrix and the input signal in the anterior layer is the vector $\mathbf{y}_k$. The only difference is that the activation functions are needed for computing the output. Thus, there can be two parallel computing structures; one is the inner product and the other is the outer product methods [17]. In the inner product method, all the input whose size can be 1,024 in this design are applied to compute one output. This results in 1,024 clock cycles to compute all the output. This design requires one very complex processing unit that can conduct the inner product of 1,024 input data at each clock. This design is not much desired because the delay of the inner product network is proportional to $\log_2(\sharp$ of input size).

We use the outer product approach that equips 1,024 simple processing units and apply one input data at each clock. Each processing unit is very simple and contains only one accumulator. This approach needs 1,024 clocks to apply all the input data sequentially. We can modify this basic outer product architecture according to the needed throughput. In order to lower the throughput, the number of processing units can be reduced, and each processing element is used in the time-multiplexed manner to compute multiple output data. Note that reducing the number of processing units lowers the throughput and may save the chip area for implementing the processing elements, but the on-chip memory size for storing the weights is not changed. Further increasing the throughput is also possible by modifying the processing elements so that they can process multiple input data at a time. If N data are provided at each clock cycle, and each processing element is modified to conduct the inner product of N weighted input data, the total number of clock cycles for each layer can be reduced to $\lceil 1024/N \rceil$.

Figure 2(a) shows the overall architecture that contains five layers. The computation in each layer is pipelined so that the maximum throughput is independent of the number of layers. Figure 2(b) illustrates the structure of a tile that updates the output values in the hidden or output layer, which computes (1) and (2). Each tile contains 16 1 k $\times$ 128-bit single-port memories (Wmem) for storing one mega weights, 1,024 processing units (PUs), and an activation function unit. Note that because we use the weights of +1, 0, and -1 in the operation of a PU instead of +Δ, 0, and -Δ, each weight is represented in two bits, and the quantization step, Δ, needs to be multiplied to a net value $\mathbf{net}_{k+1,i}$ before applying the activation function. Each tile also contains an unsigned 7-bit register that keeps the quantization step size Δ. Figure 2(c) shows the structure of a PU. The input data width, w_D, is 8 bits in the input layer and 4 bits in the hidden layers because the input precision of a phoneme recognizer needs to be high to yield good performance. Because biases need to have high precision, the bias value is represented in 17 and 13 bits in the input and hidden layers, respectively.

The timing diagram of the phoneme recognition VLSI is shown in Fig. 3. The first step of the procedure is to initialize 16 SRAMs (Wmem$_i$) with one mega weights. Because 64 weights are loaded

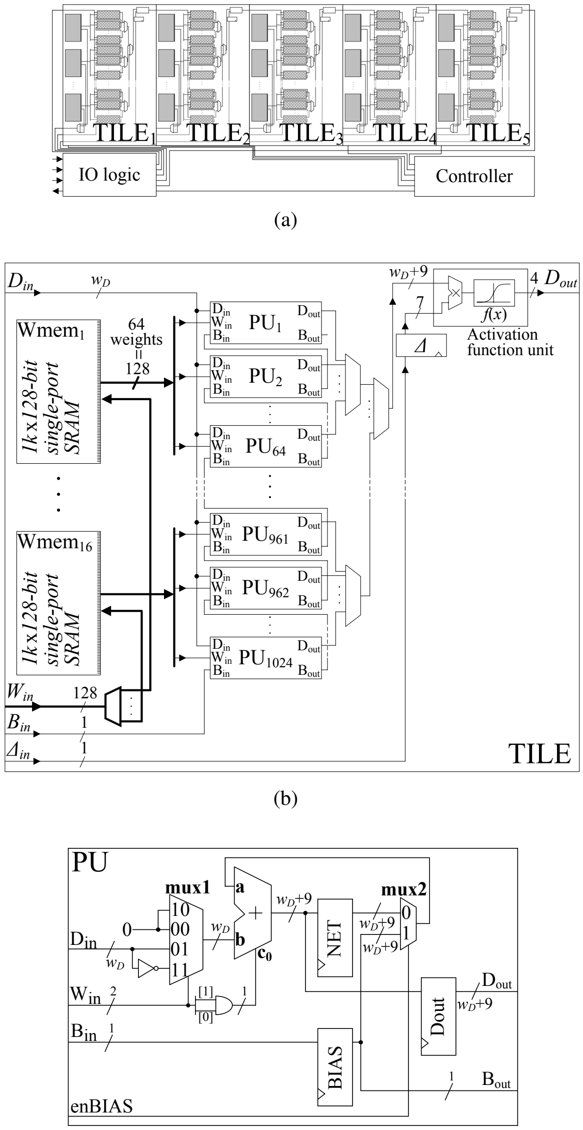

Fig. 2: Architecture of the x 1000 real-time phoneme recognition VLSI. (a) Overall architecture. (b) Structure of a tile. (c) Structure of a processing unit.

to the tile at every clock cycle, it takes 80 k clock cycles. In order to reduce the number of input ports for this phoneme recognizer, the (w_D+9)-bit bias register in a PU and the 7-bit Δ register are implemented with shift registers. Once the initialization phase is finished, the one output value in the anterior layer is applied to the tile, and then the PEs conduct outer product operations. After 1,024 clock cycles, the updated net value $\mathbf{net}_{k+1,i}$ in the i-th PU is transferred to the next tile through the activation function unit at the i-th clock cycle.

The PU in Fig. 2(c) operates as follows. In the beginning of every update operation, the (w_D+9)-bit net register, NET in Fig. 2(c), is set to zero, and the (w_D+9)-bit bias, BIAS in Fig. 2(c), is selected as the input to the adder by the multiplexer **mux2**. The multiplexer **mux1** chooses the another input to the adder according to the 2-bit

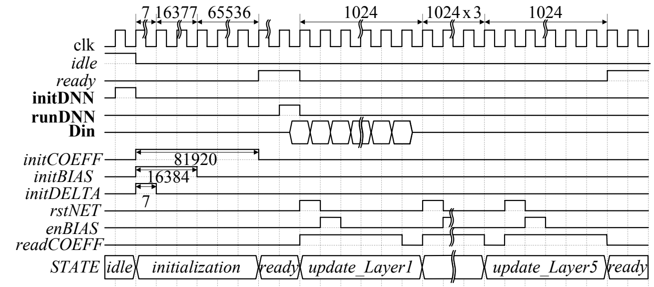

Fig. 3: Timing diagram of the x1000 real-time phoneme recognition VLSI for a frame. The input signals are shown in bold type, whereas the output and internal ones are represented in italic font.

weight W_{in}. D_{in}, 0, and $-D_{in}$ are selected according to the weight +1 (01), 0 (00), and -1 (11), respectively. The net register is updated with the output of the adder. In the subsequent 1022 clock cycles (the second to 1023-th clock cycles), the net register is chosen as the input to the adder by the multiplexer **mux2**, and the output of the adder is stored to the net register. Finally, in the last clock cycle (the 1024-th clock cycle), the output register, Dout in Fig. 2(c), is updated with the output of the adder. The output register keeps its value until the next 1024-th clock cycle, while the net register continues to update its value.

Since there are approximately 35 % of non-zero and 22 % of negative weights in the 1,024 $\times$ 1,024 weight matrix $\mathbf{W}_k$, the wordlength of the net register needs to be larger than that of the input signal D_{in} by 9 bits to prevent overflow. The activation function has a multiplier for multiplying with the quantization step size, Δ. A simple combinational logic circuit is used to convert a signed (w_D+9+7)-bit $\Delta \times \mathbf{net}_{k+1,i}$ to an unsigned 4-bit signal value. The output is pipelined to reduce the path delay.

4. EXPERIMENTAL RESULTS

The proposed feed-forward DNN VLSI with the parallel factor of 1,024 was synthesized in 0.13-μm CMOS technology using Synopsys Design Compiler. Figure 4 show the cell area for the phoneme recognition VLSI. To investigate the effect of the parallel factor, we also designed two VLSIs with the parallel factors of 64 and 256, and all the designs were synthesized under the same operating conditions.

The areas of SRAMs and registers remain the same regardless of the parallel factor, whereas that of PEs is almost in proportional to the parallel factor. This is because there are the same numbers of coefficient, bias, net, and delta memories regardless of the parallel factor. We note again that coefficients are stored in SRAMs, whereas biases, nets, and deltas are kept in registers. The area of miscellaneous logics for the parallel factor of 64 and 256 is larger than that of 1,024, because the time multiplexing operation for the parallel factor of 64 and 256 requires additional control logics. Therefore, the total cell area is not much affected by the parallel factor.

The power consumption of the phoneme recognition VLSI designs are shown in Fig. 5, which was estimated using Synopsys PrimeTime, where VLSI-1, VLSI-2, and VLSI-3 employ the parallel factors of 1,024, 256, and 64, respectively. We also operate VLSI-1 and VLSI-2 with a low frequency for a reduced throughput operation. In this case, VLSI-1 operates at the clock frequency of 25.6 MHz and 6.4 MHz for the throughput of x250 and x62.5, respectively, and VLSI-2 is clocked with the frequency of 25.6 MHz

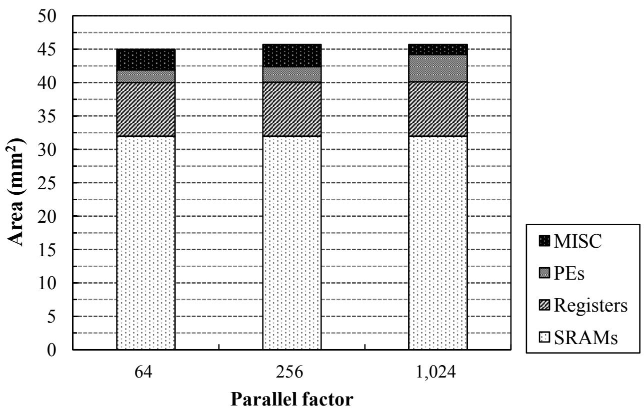

Fig. 4: Cell area of the phoneme recognition VLSI designs for the parallel factors of 64, 256, and 1,024, which performs x62.5, x250, and x1000 real-time recognition at a clock frequency of 102.4 MHz.

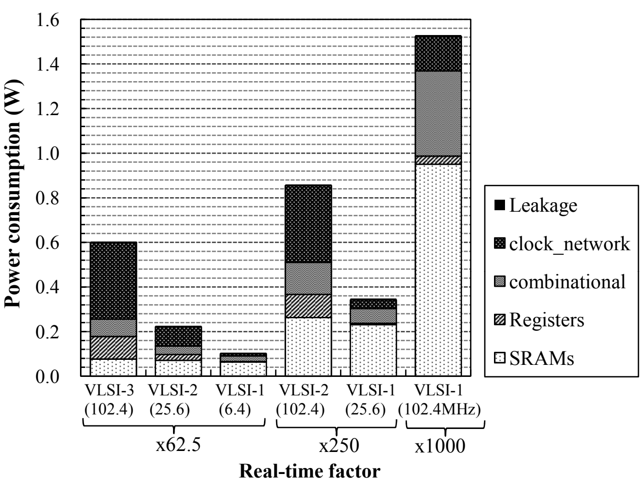

Fig. 5: Power consumption of the phoneme recognition VLSI designs for the parallel factors of 1,024, 256, and 64, which corresponds to VLSI-1, VLSI-2, and VLSI-3, respectively.

for x62.5 real-time operation. The VLSI-1 that employs the highest parallel factor is very attractive for low throughput also. The power consumption of this design is 2.1 and 5.7 times smaller than those of VLSI-2 and VLSI-3, respectively. This is because the dynamic power consumption is linearly proportional to the clock frequency.

5. CONCLUDING REMARKS

We have developed VLSI architecture for deep neural network (DNN)-based phoneme recognition. The developed feed-forward DNN employs 4 hidden layers, and each layer contains 1,024 units, and as a result the network demands a total of 5 million weights. Since each weight is represented in two bits through fixed-point optimization, the total internal memory size for storing the weights is reduced to only 10 M bits. The chip employs the parallel factor of 1,024 by equipping 1,024 low-complexity processing units in each layer. The chip occupies approximately 45.7 mm^2 with a 0.13-μm CMOS technology. The chip consumes about 1.5 W when operating at 102.4 MHz for the throughput of x1000 real-time. We also compare the designs with the parallel factors of 1,024, 256 and 64. The designs with the lowered parallel factor do not show significant chip area reduction, while consuming more power when compared to the design with a high parallel factor for obtaining the same throughput. The chip area is mostly occupied by the memory for storing weights and registers, which consumes almost the same circuit size regardless of the parallel factor. This study shows that digital DNN VLSI architecture is very advantageous for implementing very high-throughput and low-power recognition system design.

6. REFERENCES

[1] G.E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, July 2006.

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

[3] V. Vanhoucke, A. Senior, and M.Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[4] A. Mohamed, G.E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2012.

[5] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'90)*, June 1990, vol. 2, pp. 537–544.

[6] M.S. Tomlinson Jr., D.J. Walker, and M.A. Sivilotti, "A digital neural network architecture for VLSI," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'90)*, June 1990, vol. 2, pp. 545–550.

[7] C.Z. Tang and H.K. Kwan, "Multilayer feedforward neural networks with single powers-of-two weights," *IEEE Trans. Signal Process.*, vol. 41, no. 8, pp. 2724–2727, Aug. 1993.

[8] J.L. Ayala, A.G. Lomena, M. Lopez-Vallejo, and A. Fernandez, "Design of a pipelined hardware architecture for real-time neural network computations," in *Proc. IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS'02)*, Aug. 2002, vol. 1, pp. 419–422.

[9] S. Jung and S.S. Kim, "Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 265–271, Feb. 2007.

[10] 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.

[11] I.C. Goknar, M. Yildiz, S. Minaei, and E. Deniz, "Neural CMOS-integrated circuit and its application to data classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 5, pp. 717–724, May 2012.

[12] W. Sung and K.-I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Process.*, vol. 43, no. 12, pp. 3087–3090, Dec. 1995.

[13] E. Fiesler, A. Choudry, and H.J. 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.

[14] G.E. Dahl, T.N. Sainath, and G.E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Proc., (ICASSP'13)*, May 2013, pp. 8609–8613.

[15] M. Schuster and K.K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[16] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 37, no. 11, pp. 1641–1648, Nov. 1989.

[17] G.H. Golub and C.F. Van Loan, *Matrix computations*, vol. 3, JHU Press, 2012.