Efficient Hardware Implementation of Artificial Neural Networks Using Approximate Multiply-Accumulate Blocks

Mohammadreza Esmali Nojehdeh, Levent Aksoy and Mustafa Altun
Department of Electronics and Communication Engineering, Istanbul Technical University
34469, Maslak, Istanbul, Turkey
Email: {nojehdeh, aksoyl, altunmus}@itu.edu.tr

***Abstract*—In this paper, we explore efficient hardware implementation of feedforward artificial neural networks (ANNs) using approximate adders and multipliers. We also introduce an approximate multiplier with a simple structure leading to a considerable reduction in the ANN hardware complexity. Due to a large area requirement in a parallel architecture, the ANNs are implemented under the time-multiplexed architecture where computing resources are re-used in the multiply-accumulate (MAC) blocks. The efficient hardware implementation of ANNs is realized by replacing the exact adders and multipliers in the MAC blocks by the approximate ones taking into account the hardware accuracy. Experimental results show that the ANNs designed using the proposed approximate multiplier have smaller area and consume less energy than those designed using previously proposed prominent approximate multipliers. It is also observed that the use of both approximate adders and multipliers yields respectively up to a 64% and 43% reduction in energy consumption and area of the ANN design with a slight decrease in the hardware accuracy when compared to the exact adders and multipliers.**

I. Introduction

In recent years, artificial neural networks (ANNs) have achieved a remarkable performance in different research areas, including medical image processing [1], face detection [2], and semantic segmentation [3]. Recent developments in graphics processing units (GPUs) and central processing units (CPUs) provide generous memory resources and high computation speeds for training and operation of ANNs. However, for portable devices, due to their limited memory, the number of processing units, and the battery capacity, the realization of ANNs in these devices is impractical. Here, the main concern is to reduce the ANN hardware complexity taking into account the hardware accuracy.

Fig. 1(a) presents the fundamental block of ANN, *i.e.*, neuron, which sums the multiplication of input variables by weights, adds the bias value to this summation, and propagates this result to the activation function. The purpose of the activation function is to bound the amplitude of the neuron output. In mathematical terms, the neuron is described as $y = \sum_{i=1}^{n} \omega_i x_i$ and $z = \phi(y + b)$ where n denotes the number of inputs and weights. Fig. 1(b) presents an ANN design including hidden and output layers where each circle denotes a neuron.

Observe from Fig. 1 that adders and multipliers are frequently used in ANNs and dominate the hardware complexity. To reduce the ANN design area, taking into account an increase in latency, ANNs can be designed under the time-multiplexed architecture using multiply-accumulate (MAC) blocks. Fig. 2 shows a MAC-based realization of the neuron computation given in Fig. 1(a), re-using the multiplication and addition operations. In this figure, clock and reset signals are omitted for the sake of clarity. Observe that the multiplication of a weight by an input variable is realized at a time synchronized by the control block, which is actually a counter, and is added to the accumulated value stored in the register R. Under this architecture, the neuron computation is obtained after $n + 1$ clock cycles.

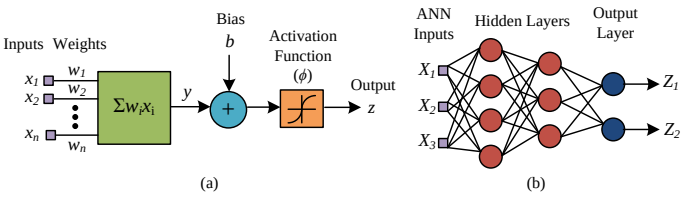

Fig. 1. (a) Artificial neuron; (b) ANN with two hidden layers.

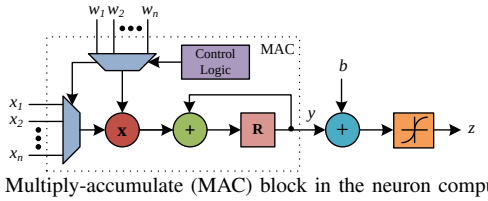

Fig. 2. Multiply-accumulate (MAC) block in the neuron computation.

Over the years, many efficient algorithms have been proposed for the reduction of ANN hardware complexity [4]–[9]. In this paper, ANNs are implemented under two different architectures using MAC blocks to explore the area and latency tradeoff. In the first one, called SMAC_NEURON, a single MAC is used to realize each neuron computation in each layer and in the second one, called SMAC_ANN, a single MAC is used to implement the whole ANN. Moreover, we present efficient hardware implementation of ANNs under the time-multiplexed architectures using approximate adders and multipliers taking into account the ANN hardware accuracy. To do so, the exact adders and multipliers in the MAC blocks are replaced by the approximate ones. Furthermore, we introduce an approximate multiplier where the tradeoff between the hardware complexity and error at the multiplier output can be explored by changing its approximation level. We note that the generation of an approximate multiplier with different bitwidths of inputs under the given approximation level can be

done in linear time as opposed to the methods of [10], [11]. Thus, as shown in [12], the ANN hardware complexity can be significantly reduced by using approximate multipliers with different approximation levels for the neuron computations at different layers. Experimental results indicate that the ANNs including the proposed approximate multiplier occupy less area and consume less energy than those including previously proposed approximate multipliers [11], [13]. It is also shown that the ANN hardware complexity can be further reduced using approximate adders.

The rest of this paper is organized as follows. Background concepts and related work are given in Section II. Section III presents the MAC-based design architectures. In Section IV, the implementation of approximate adders and multipliers used in this work are described and an approximate multiplier is introduced. Section V presents the experimental results and finally, Section VI concludes the paper.

II. Background

A. ANN Structure

An ANN is comprised of a network of neurons which are connected to each other. The weight and bias values of ANN are determined in a training phase where the error between the desired and actual response is reduced using an iterative optimization algorithm. During training, inputs are generally normalized between -1 and 1. Such a normalization may decrease the training run-time and yield an ANN with a less number of neurons and layers when compared to the ANN trained with un-normalized inputs, both achieving a similar accuracy. Furthermore, the test data are used to provide an unbiased evaluation of the final model after the training process and the accuracy, or misclassification rate, is computed as a performance metric [14].

B. ANN Implementation

To reduce the ANN hardware complexity, in [4], [5], it is shown that the weights of ANNs can be determined to include a small number of non-zero digits in training and hence, their multiplications by input variables can be realized using a small number of adders and subtractors. The floating-point weights in each layer are quantized dynamically and the fixed-point weights are expressed in binary representation in [6].

To reduce the high latency of the MAC block, a delay-efficient structure, which uses accumulators and carry-save adders, is introduced in [15]. Efficient implementation of ANN designs using MAC blocks on FPGAs is presented in [15]. Recently, MAC blocks have been used in the realization of neuromorphic cores using two models, namely axonal-based and dendritic-based [16]. A post-training method and a multiplierless design technique, that can reduce the design complexity of a time-multiplexed ANN, are given in [17].

C. Approximate Adders and Multipliers

Approximate computing refers to a class of methods that relax the requirement of exact equivalence between the specification and implementation of a computing system [18]. This relaxation allows trading the accuracy of numerical outputs for reductions in area, delay, or power dissipation of the design [19], [20]. Due to a high error-tolerance in ANNs, the use of approximate adders and multipliers in ANNs is an alternative way for the reduction of hardware complexity [9].

In [21], [22], at transistor level, approximate 1-bit adders are derived from the conventional mirror adders and XOR/XNOR based adders by removing transistors and/or replacing some parts of the adders with a small circuitry and then, a generic approximate adder is implemented using approximate 1-bit adders. In [10], [11], at gate level, design tools are generated to develop efficient approximate adders. Motivated by the drawbacks of approximation methods at transistor and gate level, a systematic synthesis technique based on a new error calculation method is introduced in [13].

In [11], a cartesian genetic programming (CGP) method is used to generate approximate multipliers. The deliberately designed approximate multiplier (DDAM) of [9] is obtained through simplifications in the truth table of the multiplication operation. A novel approximate multiplier based on the input probabilities of 1-bit adders is proposed in [13].

III. MAC-based ANN Design

Since the floating-point multiplication and addition operations occupy larger area and consume more energy than their integer counterparts [23], after the floating-point weight and bias values are found in the training phase, they are converted to integers. This conversion is simply done by multiplying each floating-point weight and bias value by 2^q, where q denotes the quantization value, and finding the least integer greater than or equal to this multiplication result.

In following, the SMAC_NEURON and SMAC_ANN design architectures are described in detail.

1) SMAC_NEURON *Architecture:* Fig. 3 presents the neuron computations at the k^{th} layer of an ANN using m MAC blocks and a common control block where m and n denote the number of outputs (or neurons) and inputs at this layer, respectively. The control block synchronizes the multiplication of input variables by the associated weights. Assuming that an ANN includes η_i neurons at each layer, where $1 \leq i \leq \lambda$ and λ denotes the number of layers, the required number of MAC blocks is $\sum_i^{\lambda} \eta_i$, *i.e.*, the total number of neurons. Note that the complexity of operations and registers in the MAC blocks are determined by the number of inputs and outputs at each layer and the weight values related to each neuron of each layer. The complexity of the control block is determined by the number of inputs at each layer. Since the neuron computations are obtained layer by layer, the neuron computations in the latter layer are started after the ones in the former layer are finished. This is simply done by generating an output signal at each layer indicating that all the neuron computations are obtained, which also disables the hardware to do unnecessary computations and enables us to reduce the power dissipation. The computation of whole ANN with λ layers and ι_i inputs at each layer, where $1 \leq i \leq \lambda$, is obtained after $\sum_i^{\lambda}(\iota_i + 1)$ clock cycles.

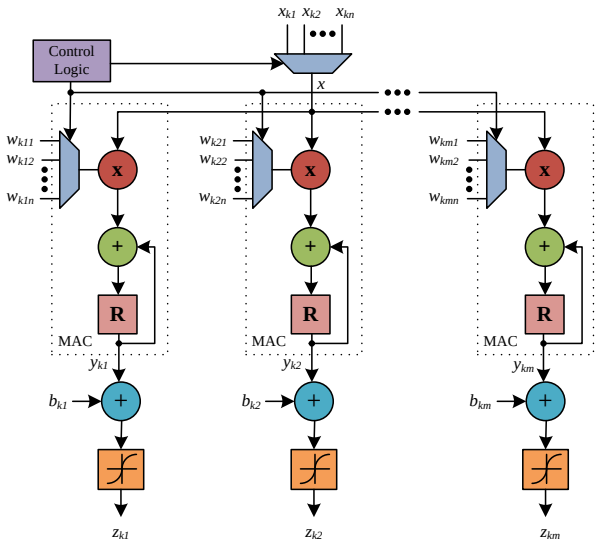

Fig. 3. Neuron computations at the k^{th} layer of ANN using MAC blocks.

2) SMAC_ANN *Architecture:* Fig. 4 shows the ANN design using a single MAC block where clock and reset signals are omitted for the sake of clarity. In this figure, the control block includes three counters to synchronize the multiplication of a weight by an input variable, the addition of a bias value to each inner product, and the application of the activation function. These counters are associated with the number of layers, number of inputs at each layer, and number of outputs (or neurons) at each layer. Note that the variables $X_1, X_2, \ldots, X_n$ denote the primary inputs of ANN and these variables are multiplied by the related weights during the computations at the first hidden layer. While the size of multiplexers for the input variables is determined by the maximum number of inputs at all layers, the size of multiplexers for the weight and bias values are defined by the total number of weight and bias values, respectively. In the MAC block, the size of multiplier is determined by the maximum bitwidth of all input variables and weights. The sizes of adder and register are defined by the maximum bitwidth of the multiplication of weights by input variables in the whole ANN. Moreover, the number of registers used to store the outputs at each layer is determined by the maximum number of outputs at each layer. We note that the computation of whole ANN with λ layers, ι_i inputs at each layer, and η_i neurons at each layer, where $1 \leq i \leq \lambda$, is obtained after $\sum_i^{\lambda}(\iota_i + 2)\eta_i$ clock cycles.

IV. Implementation of ANNs Using Approximate Arithmetic Units

In this section, we present the approximate adder of [13] and multipliers of [11], [13] used in the ANN designs, introduce an approximate multiplier and describe the implementation of ANNs using approximate adders and multipliers.

A. Approximate Adder

Fig. 5 illustrates an n-bit ripple carry adder which consists of n 1-bit full adders (FAs). In this figure, *A*, *B*, and carry-in (*Cin*) represent the input bits of FA and *Sum* and carry-out (*Cout*) denote its output bits. The truth table of 1-bit FA

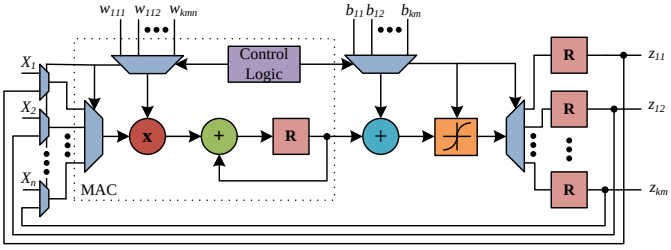

Fig. 4. ANN design using a single MAC block.

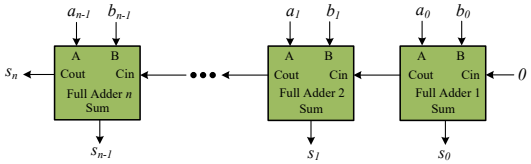

Fig. 5. Ripple carry adder.

is given in Table I. In the related studies on approximate ripple carry adders [10], [21], it is assumed that simultaneous errors on both *Sum* and *Cout* outputs of FA may generate a larger erroneous result on the adder output than an error on a single output. However, this assumption neglects the fact that while an error on one output of an FA block increases the error at the adder output, another error on the other output may decrease the error at the adder output. For example, as given in Table I, on the entry of $ABCin = 010$ for the approximate 1-bit adder APAD1, both errors on the outputs of FA generate only an error with a magnitude of 1. Thus, alternating errors on both the *Sum* and *Cout* outputs can provide an opportunity to simplify the hardware complexity of an approximate 1-bit adder. Based on this fact, 4 approximate 1-bit adders (APADs) with different error values and hardware complexity are introduced in [13]. The truth tables of these APADs are given in Table I. To obtain an n-bit approximate ripple carry adder, a synthesis method, that replaces the exact FAs by APADs with different approximation levels under the given error value, is also presented in [13].

B. Approximate Multipliers

The implementation of an exact multiplier consists of two stages, *i.e*, partial product generation using AND gates and accumulation of these partial products using half adders (HAs)[1] and FAs. An exact 4-bit unsigned multiplier structure is shown in Fig. 6(a) where rectangular blocks with 2 and 3 entries denote an HA and FA, respectively. In the design of an approximate multiplier, based on the probability of occurrences of logic 0 and 1 at the outputs of each HA and FA, the synthesis tool of [13] replaces exact HA and FA blocks in the exact multiplier by their approximate versions taking into account the error at the multiplier output and generates approximate multipliers called probability based approximate multipliers (PBAM). Also, the CGP method of [11] generates approximate multipliers which are derived from the exact multipliers.

In addition to these approximate multipliers, we propose another one, called LEBZAM, which is implemented by

[1]Half adder is obtained when one of the inputs of FA is set to 0.

TABLE I
TRUTH TABLES OF EXACT AND APPROXIMATE 1-BIT ADDERS.

Inputs			FA			APAD1				APAD2				APAD3				APAD4			
A	B	Cin	Cout	Sum	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal
0	0	0	0	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0
0	0	1	0	1	1	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1	0✓	0✗	-1	0
0	1	0	0	1	1	1✗	0✗	+1	2	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1
0	1	1	1	0	2	1✓	0✓	0	2	0✗	1✗	-1	1	0✗	1✗	-1	1	0✗	1✗	-1	1
1	0	0	0	1	1	0✓	1✓	0	1	1✗	0✗	+1	2	1 ✗	0✗	+1	2	1✗	0✗	+1	2
1	0	1	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2
1	1	0	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	1✗	+1	3	1✓	1✗	+1	3
1	1	1	1	1	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3

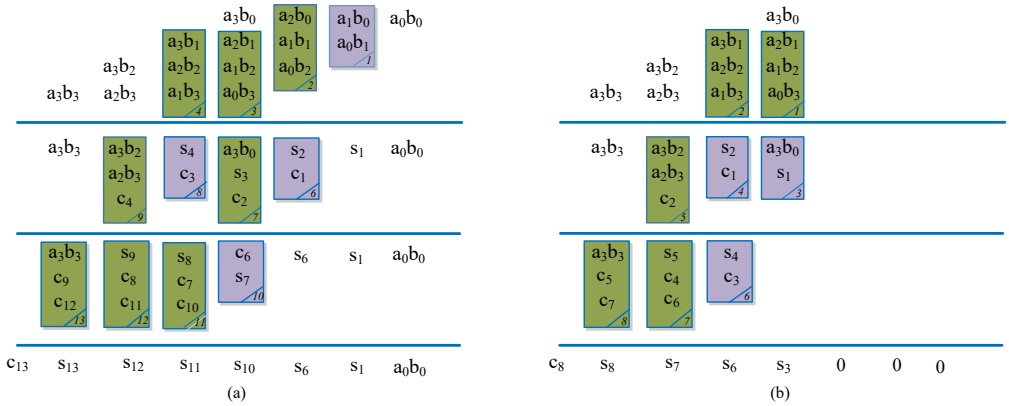

Fig. 6. (a) Exact 4-bit unsigned multiplier; (b) Approximate 4-bit unsigned multiplier with the least significant 3 bits are set to logic value 0.

setting r least significant outputs of an exact multiplier to zero, where r denotes its approximation level. The synthesis method is described as follows: i) set r least significant outputs of the exact multiplier to 0; ii) eliminate all the FA and HA blocks required to realize r least significant outputs of the exact multiplier. Fig. 6(b) illustrates the realization of 4-bit approximate multiplier when r is 3.

We note that given the approximation level and the bitwidths of the inputs, an approximate multiplier LEBZAM can be easily obtained as opposed to the approximate multipliers of [10], [11]. Thus, by using approximate multipliers with different sizes and approximation levels in the MAC blocks of ANN designs under the architectures presented in Section III, a significant reduction in the ANN hardware complexity can be achieved taking into account the hardware accuracy. Similarly, by using approximate adders of [13], the ANN hardware complexity can be further reduced.

V. EXPERIMENTAL RESULTS

As an application, we considered the pen-digit handwritten digit recognition problem [24]. In the convolution neural network design of this application, we implemented a feedforward ANN which has 16 inputs, a hidden layer with 16 neurons, and a output layer with 10 neurons. The activation functions in the hidden and output layer were symmetric saturating linear and softmax, respectively. The ANN was trained using the deep learning toolbox of MATLAB [25] where the training and test inputs were normalized in between -1 and 1, the weights were initialized randomly, and they were adjusted to minimize the error in between the actual and desired response using a backpropagation-based learning method. The ANN was trained using 7494 data and tested using 3498 data. After the training, the misclassification rate was computed as 4.85%.

After the floating-point weight and bias values were converted to integers when the quantization value q was set to 8, the ANN design using exact adders and multipliers was described in a behavioral fashion and the hardware misclassification rate (HMR) was found as 5%. In this study, the ANN designs were implemented using approximate adders and multipliers without exceeding the HMR limit which was set to 5.5%. The ANNs were implemented under the SMAC_NEURON and SMAC_ANN architectures using the approximate adders of [13], the approximate multipliers of [11], [13] and our approximate multiplier LEBZAM. The approximate multipliers of [11], *mul12s_2NM* and *mul12s_2KM*, have 12-bit inputs and are respectively chosen for their minimum area and error among other multipliers. Note that we systematically determined the approximation levels of adders and multipliers on the hidden and output layers taking into account the HMR limit value and presented the results of ANN designs with promising hardware complexity and accuracy values. The ANN designs were described in Verilog and synthesized using the Cadence Genus tool with the TSMC 40nm design library.

Tables II-V present the gate-level results of ANN designs where *area*, *delay*, and *power* stand respectively for total area in μm^2, the delay in the critical path which is determined to be the clock period in ns, and total power dissipation in mW. Also, *latency* denotes the time in ns required for the ANN

TABLE II
RESULTS OF SMAC_NEURON ARCHITECTURE USING APPROXIMATE MULTIPLIERS.

Multiplier Type	Approximation Level Hidden	Output	area	delay	latency	power	energy	HMR	area gain	energy gain
Behavioral	0	0	15327	3.58	121.68	1.44	174.77	5.00	0%	0%
$mul12s\_2NM$ [11]	NA	NA	13929	3.72	126.31	1.23	155.04	5.12	9%	11%
$mul12s\_2KM$ [11]	NA	NA	17227	3.70	125.80	1.44	181.33	5.00	-12%	-3%
PBAM [13]	7	11	13276	3.57	121.35	1.31	159.14	4.84	9%	13%
PBAM [13]	7	12	12992	3.66	124.37	1.30	161.52	5.03	15%	8%
PBAM [13]	8	11	12761	3.41	115.91	1.26	145.51	5.37	17%	17%
LEBZAM	6	9	11999	3.68	125.02	1.00	125.21	5.03	22%	28%
LEBZAM	7	11	10224	3.45	117.40	1.04	122.05	4.80	33%	30%
LEBZAM	7	12	9723	3.41	116.01	0.94	109.41	5.09	37%	37%

TABLE III
RESULTS OF SMAC_NEURON ARCHITECTURE USING APPROXIMATE MULTIPLIERS AND ADDERS.

Multiplier Type	Approximation Level Hidden Mul	Hidden Add	Output Mul	Output Add	area	delay	latency	power	energy	HMR	area gain	energy gain
Behavioral	0	0	0	0	15327	3.58	121.62	1.44	174.77	5.00	0%	0%
$mul12s\_2NM$ [11]	NA	10	NA	14	11854	3.92	133.14	0.59	78.76	5.17	22%	55%
$mul12s\_2KM$ [11]	NA	9	NA	15	13133	3.95	134.30	0.69	92.48	5.34	14%	47%
PBAM [13]	7	7	12	11	10226	3.66	124.37	0.61	76.25	5.03	33%	57%
PBAM [13]	7	7	12	12	9798	3.64	123.86	0.61	75.70	5.20	33%	57%
PBAM [13]	7	7	12	13	9534	3.66	124.37	0.62	77.25	5.17	39%	56%
LEBZAM	6	10	9	13	10392	3.58	121.72	0.58	70.11	5.31	32%	60%
LEBZAM	7	12	10	13	8801	3.61	122.88	0.55	67.32	4.88	43%	61%
LEBZAM	7	11	10	14	8989	3.61	122.81	0.52	63.68	4.97	41%	64%

output to be obtained after an input is applied, determined as the multiplication of clock period by the number of clock cycles to obtain the ANN output. The number of clock cycles required to obtain the ANN output under the SMAC_NEURON and SMAC_ANN are respectively computed as 34 and 468 for our ANN. Moreover, *energy* presents the energy consumption in pJ computed as the multiplication of latency by power dissipation. We note that the clock period was improved using the retiming technique in the synthesis tool iteratively. The switching activity data required for the computation of power dissipation was generated using the test data in simulation. This test data set was also used to verify the ANN design.

Table II presents the gate-level results of ANN designs under the SMAC_NEURON architecture where only the exact multipliers in the MAC blocks are replaced by the approximate ones. Observe that since the approximate multipliers of [11] are optimized for a fixed size, the ANN designs including these multipliers may have worse area, latency, and energy consumption values than those of ANN using exact multipliers. This is also due to the fact that the logic synthesis tool uses optimized exact multipliers and adders. On the other hand, the use of approximate multipliers of [13] can reduce the ANN hardware complexity by finding the appropriate approximation levels of multipliers at the hidden and output layers. Furthermore, our approximate multiplier leads to the largest reduction in area, latency, and energy consumption. Observe that the tradeoff between hardware complexity and accuracy can be explored by simply changing the approximation level of multipliers.

Table III presents the gate-level results of ANN designs under the SMAC_NEURON architecture where both exact adders and multipliers in the MAC blocks are replaced by the approximate ones. Observe that the use of approximate adders with the approximate multipliers reduces the ANN hardware complexity significantly. The maximum gain on area and energy consumption reaches up to 43% and 64% using our approximate multipliers.

Table IV presents the gate-level results of ANN designs under the SMAC_ANN architecture where only the exact multiplier in the MAC block is replaced by the approximate one. Although there exists only one multiplier to be replaced, the proposed approximate multiplier leads to the largest gains on area and energy consumption. Moreover, in addition to the approximate multiplier, the use of approximate adder can further reduce the hardware complexity as shown in Table V.

It is also interesting to note that the use of approximate adders and multipliers can also increase the hardware accuracy as can be observed from these results.

VI. CONCLUSION

In this paper, we presented hardware efficient implementation of ANN designs under the time-multiplexed architecture using approximate adders and multipliers. We also introduced an approximate multiplier which leads to a significant reduction in area and energy consumption in the ANN design when compared to the previously proposed approximate multipliers. Experimental results clearly show that the use of approximate adders and multipliers in the ANN designs reduces the design complexity significantly decreasing the hardware accuracy slightly when compared to the ANN designs using exact adders and multipliers. As a future work, we plan to develop an algorithm that finds the most appropriate approximate adders and multipliers used to replace the exact counterparts taking into account both the hardware complexity and accuracy.

ACKNOWLDGEMENT

This work is supported by the TUBITAK-1001 projects #117E078 and #119E507 and Istanbul Technical University BAP projects #42446.

TABLE IV
RESULTS OF SMAC_ANN ARCHITECTURE USING APPROXIMATE MULTIPLIERS.

Multiplier Type	Approximation Level	area	delay	latency	power	energy	HMR	area gain	energy gain
Behavioral	0	3180	3.52	1646.42	0.35	569.33	5.00	0%	0%
$mul12s\_2NM$ [11]	NA	3278	3.72	1738.62	0.29	499.80	5.00	-3%	12%
$mul12s\_2KM$ [11]	NA	3279	3.77	1764.83	0.29	504.74	5.00	-3%	11%
PBAM [13]	0	3287	3.79	1774.19	0.29	518.38	5.00	-3%	9%
PBAM [13]	7	3194	3.76	1760.15	0.28	499.60	4.83	-1%	12%
PBAM [13]	8	3148	3.24	1518.19	0.28	431.60	5.34	2%	24%
LEBZAM	5	3189	3.69	1725.98	0.27	472.95	4.94	-2%	8%
LEBZAM	6	3152	3.68	1724.58	0.28	489.60	4.90	1%	14%
LEBZAM	7	3091	3.56	1664.68	0.27	449.89	4.80	3%	21%

TABLE V
RESULTS OF SMAC_ANN ARCHITECTURE USING APPROXIMATE MULTIPLIERS AND ADDERS.

Multiplier Type	Approximation Level		area	delay	latency	power	energy	HMR	area gain	energy gain
	Mul	Add								
Behavioral	0	0	3180	3.52	1646.42	0.35	569.33	5.00	0%	0%
$mul12s\_2NM$ [11]	NA	13	2908	3.40	1590.26	0.25	391.63	5.06	9%	31%
$mul12s\_2KM$ [11]	NA	13	3140	3.68	1721.30	0.26	451.51	5.46	1%	21%
PBAM [13]	7	10	2972	3.55	1659.53	0.26	426.62	5.03	7%	25%
PBAM [13]	8	9	2978	3.59	1679.18	0.25	421.98	5.03	6%	26%
PBAM [13]	7	11	3029	3.84	1798.52	0.25	448.54	4.66	5%	21%
LEBZAM	6	14	3046	3.53	1652.51	0.28	469.89	4.95	4%	17%
LEBZAM	7	12	3041	3.62	1692.29	0.26	440.25	4.66	4%	23%
LEBZAM	7	13	3021	3.53	1650.17	0.26	426.73	5.40	5%	25%

REFERENCES

[1] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, "Medical image classification with convolutional neural network," in *International Conference on Control Automation Robotics Vision*, 2014, pp. 844–848.

[2] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5325–5334.

[3] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *IEEE International Conference on Computer Vision*, December 2015, pp. 1520–1528.

[4] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2016, pp. 145–150.

[5] R. Ding, Z. Liu, R. D. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *Asia and South Pacific Design Automation Conference*, 2018, pp. 1–8.

[6] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *Design Automation Conference (DAC)*, 2017, pp. 28:1–28:6.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv e-prints*, 2016, arXiv:1602.02830.

[8] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *Design, Automation and Test in Europe Conference and Exhibition*, 2015, pp. 701–706.

[9] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 28, no. 2, pp. 317–328, 2020.

[10] A. Bernasconi and V. Ciriani, "2-spp approximate synthesis for error tolerant applications," in *Euromicro Conference on Digital System Design*, 2014, pp. 411–418.

[11] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2017, pp. 258–261.

[12] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2015, pp. 701–706.

[13] M. E. Nojehdeh and M. Altun, "Systematic synthesis of approximate adders and multipliers with accurate error calculations," *Integration*, vol. 70, pp. 99 – 107, 2020.

[14] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1999.

[15] N. Nedjah, R. M. da Silva, L. M. Mourelle, and M. V. C. da Silva, "Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs," *Neurocomputing*, vol. 72, no. 10, pp. 2171 – 2179, 2009.

[16] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, and D. S. Modha, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *International Joint Conference on Neural Networks (IJCNN)*, 2012, pp. 1–8.

[17] L. Aksoy, S. Parvin, M. E. Nojehdeh, and M. Altun, "Efficient time-multiplexed realization of feedforward artificial neural networks," in *International Symposium on Circuits and Systems*, 2020, accepted for publication.

[18] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *European Test Symposium*, 2013, pp. 1–6.

[19] M. Schaffner, F. Gurkaynak, A. Smolic, H. Kaeslin, and L. Benini, "An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing," in *Design Automation Conference (DAC)*, 2014, pp. 1–6.

[20] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate Signal Processing," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 75, pp. 177 – 200, 1997.

[21] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *IEEE International Conference on Nanotechnology*, 2013, pp. 690–693.

[22] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414.

[23] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference*, 2014.

[24] F. Alimoglu and E. Alpaydin, "Combining multiple representations and classifiers for pen-based handwritten digit recognition," in *International Conference on Document Analysis and Recognition*, 1997, pp. 637–640.

[25] The MathWorks Inc., *Deep Learning Toolbox*, Natick, Massachusetts, United States, 2020. [Online]. Available: https://www.mathworks.com/help/deeplearning/