IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received November 1, 2018, accepted November 25, 2018, date of publication December 21, 2018,
date of current version January 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889110

Circuit Aware Approximate System Design With Case Studies in Image Processing and Neural Networks

TUBA AYHAN, (Member, IEEE), AND MUSTAFA ALTUN
Electronics and Communication Engineering Department, Faculty of Electrical and Electronics Engineering, Istanbul Technical University, 34469 Istanbul, Turkey
Corresponding author: Tuba Ayhan (tuba.ayhan@itu.edu.tr)

This work was supported in part by TUBITAK under Grant 117E078 and in part by the European Union's Horizon 2020 Research and Innovation Program through the Marie Skłodowska-Curie Grant under Grant 691178.

ABSTRACT This paper aims to exploit approximate computing units in image processing systems and artificial neural networks. For this purpose, a general design methodology is introduced, and approximation-oriented architectures are developed for different applications. This paper proposes a method to compromise power/area efficiency of circuit-level design with accuracy supervision of system-level design. The proposed method selects approximate computational units that minimize the total computation cost, yet maintaining the ultimate performance. This is accomplished by formulating a linear programming problem, which can be solved by conventional linear programming solvers. Approximate computing units, such as multipliers, neurons, and convolution kernels, which are proposed by this paper, are suitable for power/area reduction through accuracy scaling. The formulation is demonstrated on applications in image processing, digital filters, and artificial neural networks. This way, the proposed technique and architectures are tested with different approximate computing units, as well as system-level requirement metrics, such as PSNR and classification performance.

INDEX TERMS Approximate computing, artificial neural networks, field programmable gate arrays, high-level synthesis, image processing.

I. INTRODUCTION

Approximate computing is an ever-growing approach in computation and it recently gained an importance in power/area efficient electronics. Research is mainly focused on these two areas: design and modeling of approximate computing circuits, and analysis of system components for their error resilience.

From circuit design point of view, in the past decade, there is a rapid acceleration in approximate circuit synthesis methods, ranging from logic circuits to arithmetic units such as adders and multipliers [1]–[3]. Moreover, a Verilog based framework called Axilog [4] is also presented enabling design and reuse of approximate circuits in language abstraction level. Innovations on approximate circuits empowers system designers to promote these circuits in their error resilient applications. More complex computation units such as general purpose approximate computing machines [5], and vector processors [6], [7] are also available. However, a system level study is not performed by the circuit design methods so they do not provide a system level optimization exploiting the area and power benefits of these circuits.

From system analysis point of view, a systematic framework for analysis and characterization of inherent application resilience gives an understanding on error characteristics of 12 widely used benchmarks [8]. Work by Chippa *et al.* clearly shows approximate computing is appropriate for a wide variety of applications. However, analysis of benchmarks do not particularly answer the question of building a system satisfying a certain quality, yet using the most efficient approximate circuits. Although approximation algorithms for many signal processing applications can be derived, there is a need for high level approximate system design for area or power reduction. High level systematic methods that can automatically derive approximate circuit based on the behavioral description of the system are needed. Some of the high-level synthesis methods for approximate computing are present in the literature are SALSA [2], SASIMI [9], and ABACUS [10]. These models require eliminating costly system designs, by examining signal pairs such as in SASIMI or using an iterative stochastic greedy algorithm on a tree generated from the input behavioral description such as in ABACUS. Evaluating the entire design space would be costly, but by not evaluating the best design may be missed by these design methods. In order to cover a larger piece of the design space, search-based algorithms such as evolutionary methods are also employed in digital circuit design. By evolutionary approximation methods [11]–[14], an exact system is used to generate candidate approximations by the help of generic operators. Conventional methods trying to prune computations in the system may bring additional error due to pruning. On the other hand, complexity of search-based algorithms increase with the system size.

As an alternative to these two approaches, the overall system can be profiled for error resiliency before introducing approximate circuits in. However, profiling is costly for large systems. By appropriate error metric conversions, the ultimate system error can be modeled as a linear program together with circuit cost. In order to achieve a balance between the system quality, circuit efficiency, and profiling burden, we should investigate the circuit and system level considerations together. In this work, we aim to embody the approximate circuit design methods and system analysis by the help of a generalized optimizer. An overview on approximate circuit design methods and case studies are given in Section II, followed by the proposed system-to-circuit approximate design method in Section III. We demonstrated our optimizer in Sobel edge detector, JPEG compression, multiple Convolution Kernels (CKs) and Fully Connected Networks (FCNs), all implemented on Xilinx SPARTAN6 FPGA in Section IV. Verilog implementations of approximate computation units are provided at github.com/tubaayhan/approximatesystems. Finally the paper is concluded with future work discussions in Section V.

II. BACKGROUND

A. APPROXIMATE COMPUTING CIRCUITS

In order to obtain different levels of approximation, two main approaches are followed. The first approach include voltage over-scaling methods such as [15] where over-scaling can vastly decrease quality by its impact on MSB. The second approach propose transistor level or gate level design and implementation methods. These methods include varying the bit precision of addition and multiplication, reducing the size of carry chain, and employing look up tables for pre-computed value integration etc.

Connecting the approximate computing units with different levels of precision to each other would require additional shift operation and control circuity. In order to avoid this overhead and be able to implement an approximate version of a system by just replacing the computing circuits, truncation is not considered. An approximate adder design method is to implement approximate Full Adders (FAs) either in transistor level, such as done by IMPACT [16], or gate level as [17]. Multi-bit adders built using these approximate FAs consume less power and area when compared to exact adders. Moreover, word length of the sum is not changed when approximation level of FAs are changed.

A similar approach can be followed in approximate multiplier design. Multiplication is interpreted as simplified shift and add operation in [18], so that which is approximate in nature. Furthermore, if one of the multiplicands is a constant, then multiplication process can be converted to sum of smaller multiplications. Omitting one or more small multiplications or using an approximate adder make the process approximate. In short, all approximate circuits which do not alter the word length of the sum or product can be used in this work, such as the ones given in the library of 8-bit approximate adders and multipliers available online [3]. In the case studies, examples of approximate adders, approximate multipliers and approximate constant-multipliers will be shown.

B. OVERVIEW ON CASE STUDIES

In this work, a Sobel edge detector, a Discrete Cosine Transform (DCT) module for a JPEG encoder/decoder, multiple convolutional kernels for Convolutional Neural Networks (ConvNets) and an artificial neuron for FCNs are investigated. Here, a background on these applications are briefly given.

Firstly, A classical Sobel operator [19] is considered for edge detection. The image I is convolved with two 3×3 kernels,

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1\frac{1}{6} \end{bmatrix} * I \quad \text{and } G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1\frac{1}{6} \end{bmatrix} * I. \tag{1}$$

Norm between G_x and G_y returns the edges. Non-zero coefficients of the kernels do not require a multiplier; logic shift and sign change operations are used. However, the edge detector involves 10 2-input adder for G_x and G_y calculations and one adder for norm calculation.

Secondly, JPEG encoding is based on DCT, which converts the image in spatial domain into the frequency domain. Important frequency components, whose number is R, are then used to generate the JPEG code. Therefore, by increasing R, image quality increases and the compression ratio decreases. In JPEG encoding, multiplierless DCT such as the one in [20] is preferred because the lossy compression supports inaccurate computations. A multiplierless 8-point DCT contains 16 adders. These 16 adders do not have to execute with the same accuracy, because the high frequency components are less important than the low frequency ones.

Finally, we investigate ConvNets. ConvNets are gaining importance due to their capabilities in classification and regression of big sized data, but they are computationally expensive due to having millions of parameters [21]. Training of these networks with classical CMOS ICs (CPU and GPU) lasts for days. In order to cope with these time and area

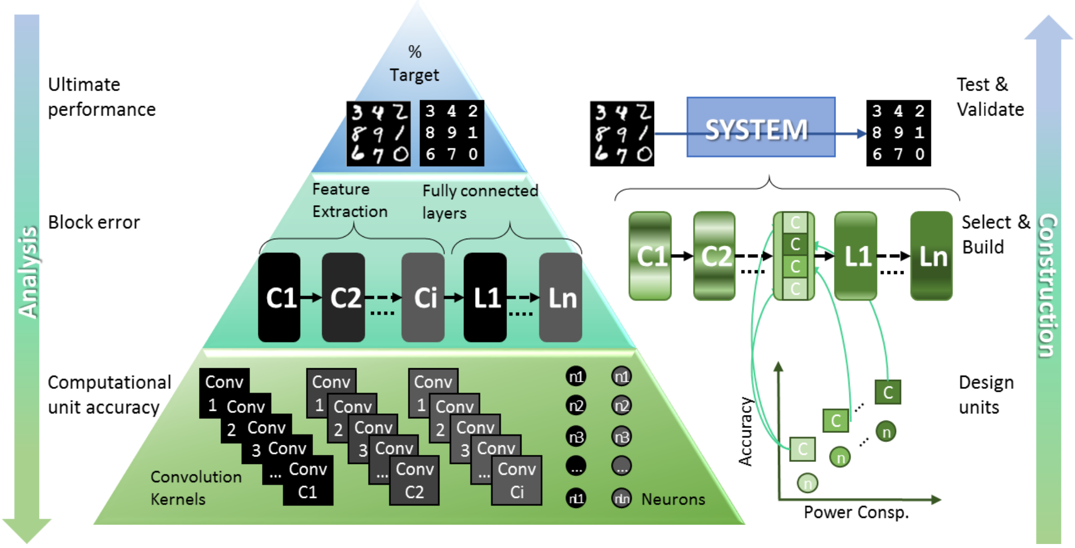


FIGURE 1. Illustration of our design methodology on ConvNets.

related problems, advanced technologies based on neuromorphic architectures and [22] and in-memory computing [23] are proposed. Moreover, high level synthesis methods such as [24] provide resource optimized convolutional neural network designs. When these methods are combined with approximate design methodologies, the logic utilization of the system can be minimized for the target system performance. So that, we applied the proposed approximate system design technique on ConvNets which are composed of two main computational parts. The first part is responsible for feature extraction with its convolution layers and the second part contains fully connected layers. Fully connected layers use artificial neurons, which as basic computation units. An artificial neuron calculates the weighted sum of its inputs and applies a nonlinear function to the sum. Although, both these weights and the coefficients in the convolution layers are trainable, training the complete ConvNet is not favored [26]. An alternative approach so called transfer learning is efficient both in accuracy and training time. In transfer learning, convolutional filter layers of a pre-trained ConvNet is adopted then fully connected layers are adapted for the target application and re-trained. In implementation, convolutional layers is more efficient to be hard-wired and fully connected layers is built in a flexible manner in order to support re-training. Therefore, a power efficient yet re-trainable ConvNet consisting of pre-trained CKs and a flexible FCN is achieved.

III. SYSTEM-TO-CIRCUIT APPROXIMATE DESIGN METHOD

The design method follows two reverse procedures: top-down analysis and bottom-up construction as illustrated in Figure 1 for ConvNets. Assume a system has N computational units, each having an error score denoted by X_i. Regardless of the ultimate performance, which can vary as PSNR, SNR, error rate etc., total computation cost of the system can be minimized by using an optimizer, which is given in Figure 2.

$$\begin{array}{ll} \textbf{Minimize:} & \sum_{i=1}^{N} P(X_i) \\ \textbf{Subject to:} & BX \leq b\ X = [X_1, X_2, ...X_N]^T \\ & X \geq LB \\ & X \leq UB \\ & X_i \in \mathbb{R}\ ,\ 1 \leq i \leq N\ . \end{array}$$

FIGURE 2. Optimization algorithm.

The optimization algorithm is meant to be run before implementing the system. It is assumed that error scores of single computation units (X_i) are available. The algorithm selects the optimum set of computation units among the availables. The optimizer is not embedded into the hardware implementation. Since the algorithm is a part of design methodology, it does not bring any area or time overhead during run time.

The variables and functions in the algorithm are explained on an accumulator example, in Figure 3. The system has one input which is denoted with S_1 and an output, S_o. The input S_1 may be noisy, with an additive white Gaussian noise identified by $\mathcal{N}(\mu_1, \sigma_1)$. Then, the SNR at the output can be calculated as a function of μ_1 and σ_1. Accumulator output is $So = \sum_{m=1}^{M} S1(m)$. If the adder circuit in the accumulator

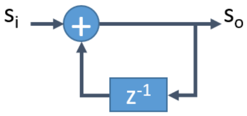


FIGURE 3. A simple accumulator example explaining the approximate design method.

block is an exact adder, the SNR at the output is

$$\mathrm{SNR}_o = \frac{M\mu_1}{\sqrt{M}\sigma_1} = \sqrt{M} \times \mathrm{SNR}_1. \tag{2}$$

The difference between the outputs of an exact and approximate accumulator is the noise added by the approximate adder. Let's model the additive noise from the adder by a normal distribution: $\mathcal{N}(\mu_a, \sigma_a)$. In order to obtain an unbiased output, the mean of the adder noise, μ_a, has to be 0. Then, using an approximate adder the output SNR is found as

$$\mathrm{SNR}_o = \sqrt{M}\frac{\mu_1}{\sqrt{\sigma_1^2 + \sigma_a^2}}. \tag{3}$$

If the input SNR is known, the approximation level of the adder can be calculated for a target output SNR, which is denoted as SNR_t.

The optimizer's objective function is dominated by the circuit requirements. The cost of a computational unit in a particular realization is a function of its error score as in $P(X_i)$. In our example, total arithmetic power consumption of the accumulator is equal to the power consumption of the adder: P_1. On exact and 6 different approximate adders of 8-bit each are implemented on FPGA and their power consumption is analyzed in Figure 4. Each approximate adder has a different error score, represented by its precision, σ_i. Therefore, they each have a different power consumption as shown by black dots in the figure. Mean error of each approximate adder is 0, since signed addition is considered in this example. Exact adder in this implementation setting consumes 2.1μW and has an error score of 0. The function $P(\cdot)$ determines the relationship between the accuracy and the cost of the arithmetic block. This function can give power consumption, transistor count, LUT count etc. and it has to be estimated within the circuit conditions. The objective function is derived using fitting these 3 types of curves to the data points: 1. linear, 2. second order polynomial and 3. rational with nominator and denominator degrees of 0 and 1. Objective function is related with the error score as

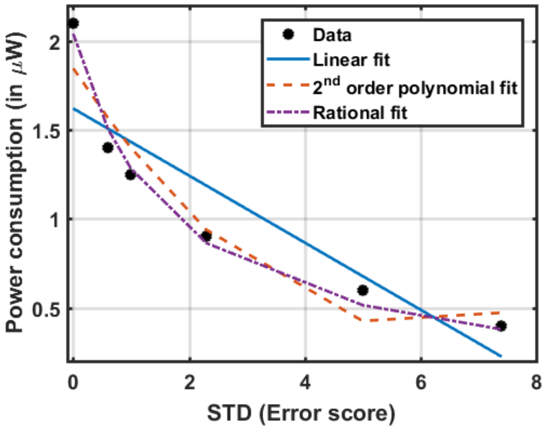


FIGURE 4. Power consumption of an exact and 6 approximate adders are plotted versus their error scores.

1) $P(i) = -0.1886\times\sigma_i + 1.621$ with a curve fitting RMSE of 0.3136,
2) $P(i) = 0.041\times\sigma_i^2 + -0.4892\times\sigma_i + 1.847$ with a curve fitting RMSE of 0.2247,
3) $P(i) = 3.443/(\sigma_i + 1.686)$ with a curve fitting RMSE of 0.07845.

The main optimizer constraint is derived from the contribution of each computational unit to the ultimate score. In the constraint $\mathbf{BX} \leq b$, b and $\mathbf{B}$ denote the maximum amount of tolerable error and the projection of error scores to the ultimate error, respectively. The variables l and u represent the lower and upper bound for error scores of the computational units. For the accumulator example, the constraint can be derived from (3). The constants M, σ_1, and μ_1 forms the matrix B. Moreover, b will become $1/\mathrm{SNR}_t$, so that $\mathrm{SNR}_o \leq \mathrm{SNR}_t$.

In order to clearly see the effect of objective function on this algorithm, assume the white noise contribution is discarded from the system, so SNR_o becomes $\frac{\sqrt{M}}{\sigma_a}$. After the algorithm is run, an optimum σ_a value is found. The adder to be implemented is selected as the one with the closest precision. In this example, all objective functions return the same approximate adder for the same target SNR. For a different application where the computation units are not very far from each other in terms of accuracy, selecting a non-linear objective function may improve the overall circuit cost reduction.

The optimization goal is to minimize the circuit cost, which can be power consumption, area or any other asset. Moreover, ultimate system performance such as MSE, SNR, PSNR, classification error etc. determines the optimizer constraint. Above explained optimization algorithm can be applied to many systems, when the circuit cost metric and ultimate system performance metric are translated into objective function and constraint, respectively. The link between objective and constraint is built through the computational circuits which allow trading off system performance with circuit cost. In the following section, case studies with different cost and performance metrics are investigated. In the showcases, a Matlab solver with dual-simplex algorithm is used. Runtime varies with the complexity of the circuit. In the case studies given in the following section, runtime of the solver on Matlab is under a second.

IV. CASE STUDIES

In the rest of the paper, we demonstrate our algorithm on three different case studies summarized in Table 1. In the optimizer, the objective function and the constraints are driven by circuit cost and the target performance of the system, respectively. Sobel edge detection and JPEG encoding cases use multiplierless convolutions and additions, so that the computational units are multiple adders for these cases. The system performance, PSNR in these cases, has to be written in terms of adder accuracies. Power consumption and area utilization of approximate adders increase with their accuracy.

TABLE 1. Summary of case studies.

Case studies	Circuit cost	Computational units	Ultimate system perf.
Sobel edge detector	Power consp.	Adder	Filtered image PSNR
JPEG encoding and decoding	Power consp.	Adder	Decoded image PSNR
ConvNet - Feature Extraction	Area (LUT count)	Multiple conv. kernels	MSE
ConvNet - Fully connected layers	Area (LUT count)	Artificial neuron	Classification rate

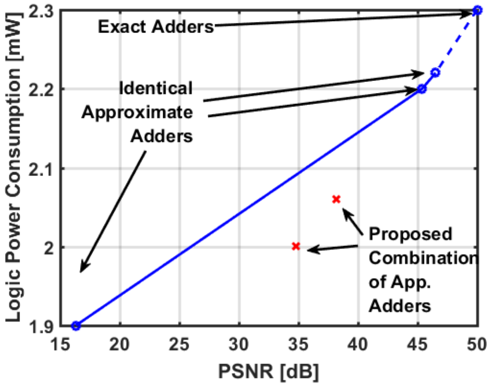


FIGURE 5. Six different approximate implementations of Sobel edge detector.

Therefore, power consumption is also derived in terms of adder accuracy. To minimize the total power consumption, we find the best combination of approximate adders as explained in [25]. For the third case study, we investigate ConvNets under two parts. Firstly, we implement multiple convolution kernels with approximate coefficients in minimum area, yet achieving a target Mean Square Error (MSE). Secondly, we minimize area of a FCN using our optimizer and approximate neurons.

A. SOBEL EDGE DETECTOR

We first derive the objective function then the constraints. Firstly, error score of an adder i is ϵ_i, which is defined as the mean of error distance between the actual and expected sums of the approximate adder. The adders are represented with $X = [X_1, X_2, \ldots X_{11}]^T$, such that each adder is one of the four different types: $Xi \in \epsilon_0, \epsilon_1, \epsilon_2, \epsilon_3$. The adder with ϵ_0 is an exact adder and it covers the largest area as well as it consumes the most power among other approximate adders. Power consumption of adders are denoted by p_i. A relationship, $p_i = m\epsilon_i + n$, between $\epsilon = [\epsilon_i]$ and $p = [p_i]$ is estimated with Least Mean Squares algorithm. As a result, the objective function is a linear function of adder errors.

The system output is an image with an ultimate performance expectation of certain PSNR which is a function of MSE, thus each pixel has equally important contribution to PSNR. Therefore, the optimizer's constraint is formed with b being the target MSE and $B = [1/S_1, 1/S_2, \ldots 1/S_{11}]$, where S_i is the output size of the adder X_i. The method is tested for 35 dB PSNR and 40 dB PSNR targets. The system is built and tested with famous images: Lena, Baboon, Cameraman and Barbara. Actual average PSNR versus logic power consumption is plotted in Figure 5. Since the edge detector requires 11 adders, and there are 4 different types of adders, there are 4^{11} possible implementations. Instead of comparing more than 4 million implementations, we compared our optimizer's solutions with standard implementations. On the blue line of Figure 5, we see the systems where all 11 adders are identical, either exact or one of the 3 approximate adders. Fully exact implementation is spotted on the upper-right corner of the figure, where both PSNR and power consumption are at their maximum. The blue line forms a border, any implementation that falls under this line is more power efficient. When the adders are selected with our optimizer, the performance of the system remains below this line. That means, our optimized power saving systems provide higher PSNR for the same amount of power in comparison to nonoptimal systems.

B. JPEG ENCODING AND DECODING

Unlike the previous case study for Sobel edge detector, each output of the approximate DCT unevenly contributes to the ultimate performance criteria, PSNR. The objective function is to minimize the total computational power consumption of the DCT module in a JPEG encoder. Moreover, the constraint b changes with the target compression ratio, or number of encoded frequency components, R. Therefore, computations to find the high frequency components are assigned a lower weight while writing the constraint equation.

In Figure 6, performance of approximate DCTs is compared with an exact DCT. In the approximate designs, maximum 1 dB PSNR loss is allowed, after decoding. In other words, quality constraint for this application is 1 dB PSNR tolerance. Since PSNR is also a function of R, the optimizer is run to find the most power efficient implementation. Exact implementation is not changed with R, so that its power consumption is stable at 17 mW, which is the maximum value in the figure. On the other hand, power consumption is decreased by approximate implementation, as shown by

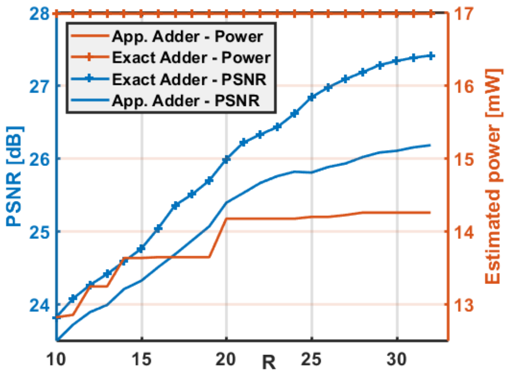
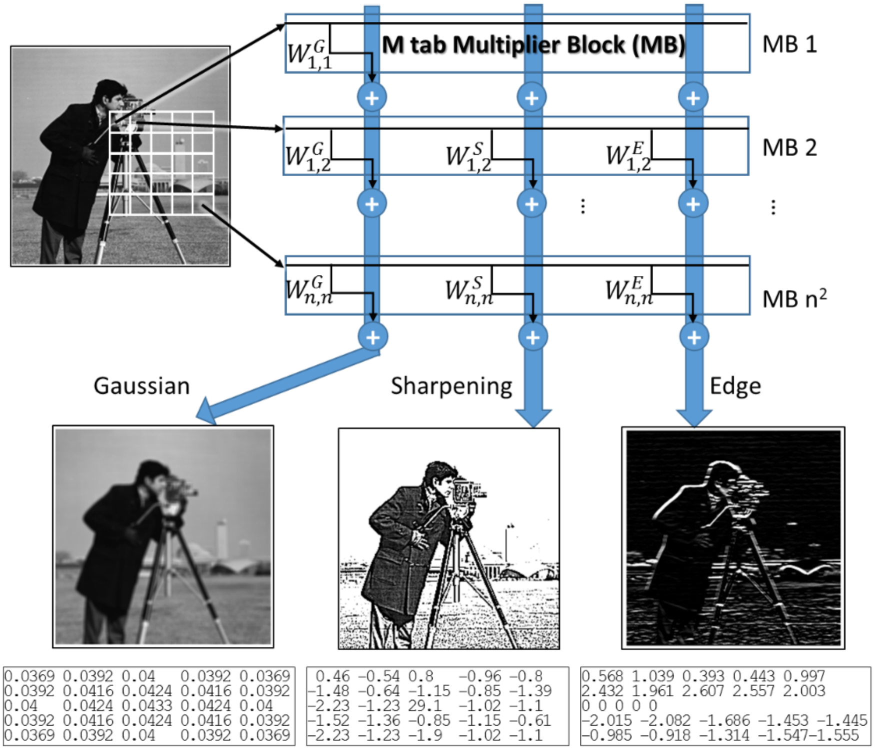


FIGURE 6. Power (red on the right axis) and PSNR (blue on the left axis) of an approximate DCT is compared with an exact DCT.



0.0369	0.0392	0.04	0.0392	0.0369
0.0392	0.0416	0.0424	0.0416	0.0392
0.04	0.0424	0.0433	0.0424	0.04
0.0392	0.0416	0.0424	0.0416	0.0392
0.0369	0.0392	0.04	0.0392	0.0369

0.46	-0.54	0.8	-0.96	-0.8
-1.48	-0.64	-1.15	-0.85	-1.39
-2.23	-1.23	29.1	-1.02	-1.1
-1.52	-1.36	-0.85	-1.15	-0.61
-2.23	-1.23	-1.9	-1.02	-1.1

0.568	1.039	0.393	0.443	0.997
2.432	1.961	2.607	2.557	2.003
0	0	0	0	0
-2.015	-2.082	-1.686	-1.453	-1.445
-0.985	-0.918	-1.314	-1.547	-1.555

FIGURE 7. M $n \times n$ convolution kernels are implemented as n^2 M-tab multiplier blocks.

with the red lines. With the given quality constraint, power consumption can be reduced by 15.8% to 25.12%. PSNR of exact and approximate implementations are given with blue lines.

C. CONVOLUTIONAL NEURAL NETWORK

We investigate convolutional kernels and fully connected network separately.

1) CK - IMPLEMENTING MULTIPLE CONVOLUTION KERNELS WITH DIFFERENT ACCURACIES

The weights of pre-trained CKs are not changed, thus they can be hardwired. As a simplified case study, we implement multiple convolution kernels processing the same image, as illustrated in Figure 7. As the frame slides over the input image, one pixel has to be multiplied with kernel coefficients of Gaussian, Sharpening, and Edge filters at position (x, y), denoted as $W^G_{(x,y)}$, $W^S_{(x,y)}$, and $W^E_{(x,y)}$, respectively. For each coefficient, we have a multiplier block (MB) where the only input is a pixel value and the output are this pixel multiplied by different constants. If we can approximate the convolution coefficients, we can minimize the total MB area. To do so, we modify multiple coefficient multiplication (MCM) optimization [27] to assign different amount of error on every coefficient in the MB.

We use same size (5×5) filters with the characteristics of Gaussian, Sharpening, and Edge filters. If we synthesize 2 different filters with 8-bit coefficients from each group, we need 2424 LUTs on a Spartan6 FPGA. In order to reduce the LUT count, approximate MBs are generated. There are n^2 MBs in Figure 7. Each of them has three constant multipliers, in this example. The proposed MBs are biased, because all input and coefficients are non-negative numbers. So that, the optimization algorithm's constraint is a function of MB mean error and the target system performance is used as filtered image mean error. In generation process, we let the filter coefficients deviate as $\Delta W_{x,y}$, where $\Delta W_{x,y}$ is in $\pm e$. Then, assuming the mean of the image and coefficient deviation are $\hat{I}$, and $\Delta\hat{W}$, respectively, the mean of the additive error a pixel is $\hat{p} = 9 \times \hat{I} \times \Delta\hat{W}$ for convolution kernels with 9 non-zero coefficients. With the above assumption, mean error on filtered image, t is equal to mean error on a pixel. The optimization algorithm selects error margins e^G, e^S, and e^E for the Gaussian, Sharpening, and Edge filter coefficients. In the optimization algorithm's constraint, $BX \leq b$, becomes

$$\left[9\times\hat{I}, 9\times\hat{I}, 9\times\hat{I}\right]^T \cdot \left[\Delta\hat{W}^G, \Delta\hat{W}^S, \Delta\hat{W}^E\right] \leq \left[t^G, t^S, t^E\right]^T \tag{4}$$

The circuit cost is derived as a function of $\Delta\hat{W}$, by using constant multiplier implementation results. In order to generate data to plot Figure 8, 8-bit constant multipliers (CMs) with coefficients $c \in [0255]$ are implemented. Logic utilization of each implementation is recorded as $Cost_c$. An approximate CM for original coefficient c_{orig} is the CM implementation

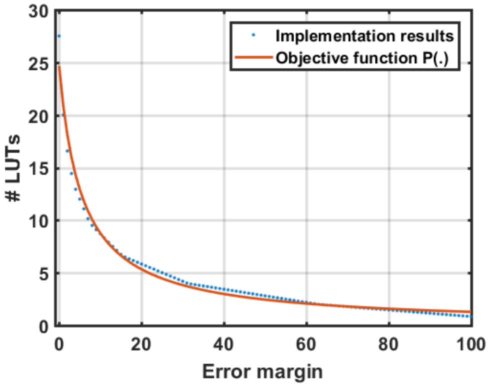
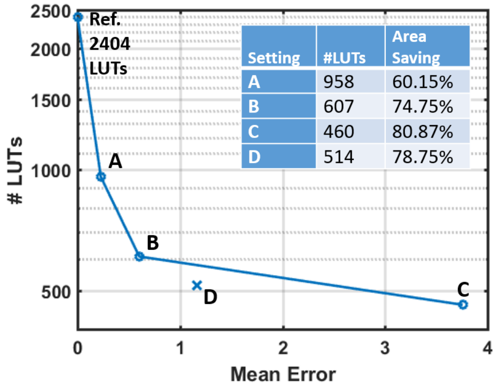


FIGURE 8. Mean area utilization of an approximate 8-bit constant multiplier with a sweep on error margin.



Setting	#LUTs	Area Saving
A	958	60.15%
B	607	74.75%
C	460	80.87%
D	514	78.75%

FIGURE 9. Area and error results of multiple CK implementation optimized with 4 different settings.

with minimum cost in its neighborhood. Thus:

$$c_{approx} = \text{argmin}_c([Cost_c]),\ c \in [c-e, c+e] \qquad (5)$$

For error margin $e \in [0\,100]$, average of CM cost over all possible coefficients (0-255) is plotter in Figure 8. Therefore, error margin e is calculated using the optimization algorithm. Then, we run MCM optimization using Settings A-D with different error margins on the coefficients and reported the corresponding mean error between the approximate filter image and the desired image per filter group in Table 2. Settings A-C assign same amount of coefficient error to different filter groups, whereas in setting D the coefficients corresponding to coefficient errors are grouped according to the filter type. We gain 60.15% to 80.87% area saving in CK implementation, as given in Figure 9. Moreover, we can reduce the area consumption without aggressively changing the filter characteristics. This is possible due to grouping the multiplicands according to their maximum tolerable error. As a result, MCM optimization with grouped errors allows us to reduce the area of convolutional layers, proving the idea that the convolutional filters are better to be grouped [28], [29].

TABLE 2. 4 settings with different error margin (e) on coefficients are tested. Performance of the system is given on each filter group as well as the overall system, per setting. error margin on settings A,B, and C are 2,4, and 8 respectively. On setting D, error margins are 2,4, and 8 for Gaussian, Sharpening, and Edge filters, respectively.

	Mean error on filtered image			
Setting	Gaussian	Sharpening	Edge	Mean over all
A	0.4206	0.0697	0.1952	0.2285
B	1.5011	0.1287	0.1790	0.6029
C	5.6418	4.1688	1.4628	3.7578
D	1.1122	1.7787	0.6042	1.165

2) FCN - ESTIMATING ERROR SCORES FROM TARGET CLASSIFICATION RATE

In this work, the neurons with 8-bit inputs, I, and 8-bit weights, W, are designed to be configured for 3 approximation grades: i. approximate adder + approximate multiplier, ii. exact adder + approximate multiplier, iii. full exact implementation. Since a multiplier consumes more power and area than adder circuits, we do not consider using exact multipliers with approximate adders. We use the same approximate adders as the ones in the previous cases. We use shift-add approach for multiplication in [18]. Shift-add approach depends on decomposition of products. In Figure 10, W is decomposed into two parts which are denoted by LSB and MSB. Each decomposition is carried on MultLevel1 block, where I is shifted by an amount that is determined by LSB or MSB part of W. MSB product is shifted by the bit length of the LSB part and the decomposed products are summed. In order to balance the computation time of FCNs with CKs as well as to provide more area for CKs, we prefer an accumulator at the output of the multiplier. The accumulator output is passed through nonlinear ReLu function as in AlexNet ; ReLu does not require any arithmetic block. We start with a restriction that each layer is composed of only one type of neuron. Therefore, the objective function is to minimize the cost of total FCN defined as $\sum_{i=1}^{L} N_i P(A_i)$, where L is the number of layers in FCN. The layer i has N_i neurons with an approximation error modeled by A_i. Cost of a neuron is $P(\cdot)$. Neurons with different amount of approximation introduced above are implemented on FPGA and power consumptions are measured by XPA tool. This way, a linear objective function is obtained.

In order to write the constraints, classification performance has to be converted into computation noise. The constraint is to keep the difference between classification error of an exact network, e_b and the ultimate classification error rate, e_t, after approximate implementation within a certain limit. In binary classification, a correct classification occurs if one output is above a certain threshold and the other output is below, ideally 1 and -1, respectively. Therefore, if the output of the neurons are close to the threshold, computation error may cause classification error. In other words, the decision boundary is based on the output distributions of neurons. Distribution of a perfectly functioning output neuron does not intersect with the distribution of others. Size of this intersection region is correlated with the error rate. We let the intersection region

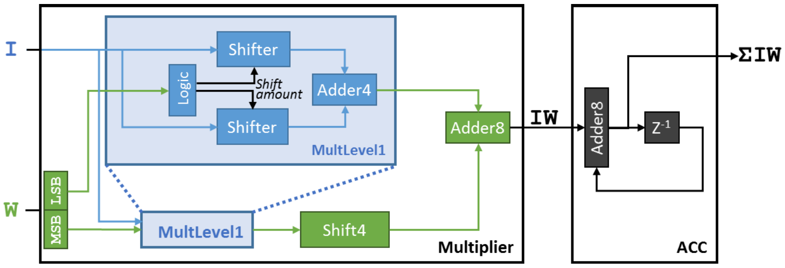


FIGURE 10. Master multiplier architecture is combined with an accumulator.

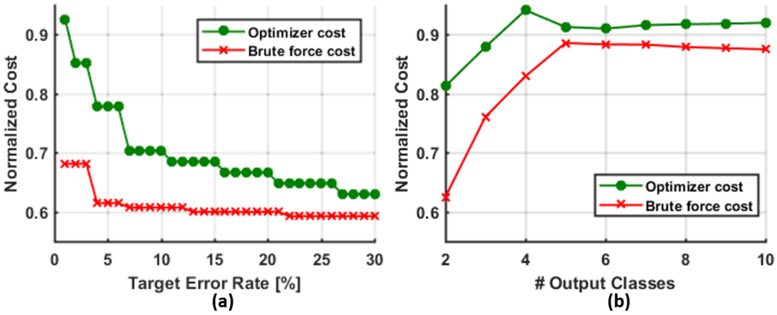


FIGURE 11. Approximate FCN optimization results: (a) 2 class Iris dataset, (b) 2-10 class MNIST dataset.

grow by adding computation noise to the neurons. A neuron in a hidden or output layer i, receives erroneous inputs, and calculate their weighted sum with a computation error. Assuming the computation error is normally distributed, total noise at the input of a neuron is averaged out by the number of its inputs. Therefore, H_{i+1}, error contribution of layer $i+1$, depends on the neuron type of layers i and $i+1$, as given with

$$H_{i+1} = A_{i+1} + \frac{A_i}{\sqrt{2N_i}}. \quad (6)$$

Error contribution of the output layer and e_b constitutes the classification error rate, et. Approximation error can vary between e_b and e_t. If the approximation error model is standard deviation, as further be used in this work, then the constraint of the model optimizer is formed as

$$H_{i+1} \leq \frac{1}{n_c} \| \sqrt{e_t - e_b} \|. \quad (7)$$

Tolerated error region is scaled inversely proportional to the number of classes, n_c. We demonstrate optimized FCN in two famous datasets: Iris and MNIST [30]. Two classes of Iris is used to train a network with 5 and 20 neurons in hidden layers. We run our optimizer varying the target error rate from 1% to 30%. Cost of an exact network is normalized to 1. The normalized cost of the proposed networks are plotted in Figure 11.a. Our optimized results occupy 8% to 37% less area than the exact network. We compared our results to brute force search results, in red line. Brute force search compares the area costs of 64 different implementations, which takes 5 hours to implement only (without testing or verification) on FPGA using Xilinx ISE tools. According to the brute-force search, 31% to 40% area reduction is possible with approximate implementation. It should be noted that our optimized networks always achieve the target classification performance.

A network with 10 and 20 neurons in hidden layers is trained for 2-10 classes in MNIST dataset. Their baseline error changes between 1.2% and 7.8%. For each network, the optimizer is run with a maximum 10% classification error constraint as well as a brute force search is done to find the most area efficient network satisfying the same constraints. As shown in Figure 11.b, our proposed FCN provides cost reduction, yet error rate is still smaller than the target. If the proposed optimization method is combined with a high level synthesis method in [24], resource utilization can be optimized in terms of both memory and logic.

V. CONCLUSION

In summary, we bring approximate circuit design methods and system analysis techniques together to implement quality scalable yet cost efficient systems. To successfully employ approximate circuits in systems of different scales, the trade-offs between circuit costs and system's output quality are modeled as a linear program. Moreover, architectures that efficiently use approximate circuits are proposed for different applications. These applications include image processing and artificial neural networks. The formulation is tested with systems used in edge detection, image compression, and ConvNets, showing the widespread applicability of the proposed technique. As a future work, a circuit aware approximate system framework assisted automated circuit synthesis would highly increase system design efficiency. Therefore, combining this method with circuit design tools is a substantial step on approximate design automation area. Moreover, many systems include various components with different cost considerations. For example, RF components of a system is usually the most power consuming part whereas the digital processor is the most area consuming one. These cost definitions should be incorporated within the optimizer to satisfy multiple design constraints on approximate SoC or NoC systems, as another future work.

REFERENCES

[1] D. Shin and S. K. Gupta, “Approximate logic synthesis for error tolerant applications,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)* Dresden, Germany, 2010, pp. 957–960.

[2] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, “SALSA: Systematic logic synthesis of approximate circuits,” in *Proc. DAC Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 796–801, doi: 10.1145/2228360.2228504.

[3] 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 *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Lausanne, Switzerland, 2017, pp. 258–261, doi: 10.23919/DATE.2017.7926993.

[4] D. Mahajan *et al.*, “Axilog: Abstractions for approximate hardware design and reuse,” *IEEE Micro*, vol. 35, no. 5, pp. 16–30, Sep./Oct. 2015.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural acceleration for general-purpose approximate programs,” *IEEE Micro*, vol. 33, no. 3, pp. 16–27, May 2013, doi: 10.1109/MM.2013.28.

[6] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Quality programmable vector processors for approximate computing,” in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Davis, CA, USA, Dec. 2013, pp. 1–12.

[7] H. Jia and N. Verma, “Exploiting approximate feature extraction via genetic programming for hardware acceleration in a heterogeneous microprocessor,” *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 1016–1027, Apr. 2018, doi: 10.1109/JSSC.2017.2787762.

[8] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, May/Jun. 2013, pp. 1–9.

[9] S. Venkataramani, K. Roy, and A. Raghunathan, “Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits,” in *Proc. Design, Autom. Test Eur. Conf. (DATE)*, Grenoble, France, 2013, pp. 1367–1372, doi: 10.7873/DATE.2013.280.

[10] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, “ABACUS: A technique for automated behavioral synthesis of approximate computing circuits,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Dresden, Germany, 2014, pp. 1–6, doi: 10.7873/DATE.2014.374.

[11] Z. Vasicek and L. Sekanina, “Evolutionary Approach to approximate digital circuits design,” *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, Jun. 2015, doi: 10.1109/TEVC.2014.2336175.

[12] Z. Vasicek and L. Sekanina, “Evolutionary design of complex approximate combinational circuits,” *Genet. Program. Evolvable Mach.*, vol. 17, no. 2, pp. 169–192, Jun. 2016, doi: 10.1007/s10710-015-9257-1.

[13] R. Hrbacek, “Parallel multi-objective evolutionary design of approximate circuits,”ÂÎ in *Proc. Genet. Evol. Comput. Conf., 17th Genet. Evol. Comput. Conf (GECCO)*, Jul. 2015, pp. 687–694, doi: 10.1145/2739480.2754785.

[14] H. Norouzi and M. E. Salehi, “Evolutionary design for energy-efficient approximate digital circuits,” *Microprocessors Microsyst.*, vol. 57, pp. 52–64, Mar. 2018, doi: 10.1016/j.micpro.2018.01.002.

[15] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “DVAFS: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Lausanne, Switzerland, 2017, pp. 488–493, doi: 10.23919/DATE.2017.7927038.

[16] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: IMPrecise adders for low-power approximate computing,” in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Fukuoka, Japan, Aug. 2011, pp. 409–414, doi: 10.1109/ISLPED.2011.5993675.

[17] S. Dutt, H. Patel, S. Nandi, and G. Trivedi, “Exploring approximate computing for yield improvement via re-design of adders for error-resilient applications,” in *Proc. 29th Int. Conf. VLSI Design, 15th Int. Conf. Embedded Syst. (VLSID)*, Kolkata, India, 2016, pp. 134–139, doi: 10.1109/VLSID.2016.101.

[18] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, “Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing,” in *Proc. Design, Autom. Test Eur. Conf. (DATE)*, Dresden, Germany, 2016, pp. 145–150.

[19] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. Hoboken, NJ, USA: Wiley, 1973, pp. 271–272.

[20] J. Liang and T. D. Tran, “Fast multiplierless approximations of the DCT with the lifting scheme,” *IEEE Trans. Signal Process.*, vol. 49, no. 12, pp. 3032–3044, Dec. 2001, doi: 10.1109/78.969511.

[21] X.-W. Chen and X. Lin “Big data deep learning: Challenges and perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014, doi: 10.1109/ACCESS.2014.2325029.

[22] Y. Wang, R. Chen, R. Mao, and Z. Shao, “Optimally removing synchronization overhead for CNNs in three-dimensional neuromorphic architecture,” *IEEE Trans. Ind. Electron.*, vol. 65, no. 11, pp. 8973–8981, Nov. 2018, doi: 10.1109/TIE.2018.2813959.

[23] Y. Wang, W. Chen, J. Yang, and T. Li, “Towards memory-efficient allocation of CNNs on processing-in-memory architecture,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1428–1441, Jun. 2018, doi: 10.1109/TPDS.2018.2791440.

[24] D. H. Noronha, B. Salehpour, and S. J. E. Wilton, “LeFlow: Enabling flexible FPGA high-level synthesis of tensorflow deep neural networks,” in *Proc. 5th Int. Workshop FPGAs Softw. Programmers FSP Workshop*, Dublin, Ireland, 2018, pp. 1–8.

[25] T. Ayhan, F. Kula, and M. Altun, “A power efficient system design methodology employing approximate arithmetic units,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Bochum, Germany, Jul. 2017, pp. 243–248, doi: 10.1109/ISVLSI.2017.50.

[26] K. Bong, S. Choi, C. Kim, and H.-J. Yoo, “Low-power convolutional neural network processor for a face-recognition system,” *IEEE Micro*, vol. 37, no. 6, pp. 30–38, Nov./Dec. 2017, doi: 10.1109/MM.2017.4241350.

[27] L. Aksoy, P. Flores, and J. Monteiro, “Approximation of multiple constant multiplications using minimum look-up tables on FPGA,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 2884–2887, doi: 10.1109/ISCAS.2015.7169289.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, Dec. 2012, pp. 1097–1105.

[29] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi. (2016). “Deep roots: Improving CNN efficiency with hierarchical filter groups.” [Online]. Available: https://arxiv.org/abs/1605.06489

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

TUBA AYHAN (M’16) received the Ph.D. degree in electrical engineering from KU Leuven, Belgium, in 2016. She is currently a Researcher with the ECC Laboratory, Istanbul Technical University. Her research interests include embedded system design and power-efficient digital circuit implementation.

MUSTAFA ALTUN received the Ph.D. degree in electrical engineering with a minor in mathematics from the University of Minnesota, in 2012. Since 2013, he has been an Assistant Professor of electrical engineering with Istanbul Technical University, where he runs the Emerging Circuits and Computation Group. He has authored more than 50 peer-reviewed papers. He was a recipient of the TUBITAK Success, TUBITAK Career, and Werner von Siemens Excellence Awards.