Received 27 July 2022, accepted 30 August 2022, date of publication 5 September 2022, date of current version 15 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204704

RESEARCH ARTICLE

Energy-Efficient Precision-Scaled CNN Implementation With Dynamic Partial Reconfiguration

EMAN YOUSSEF[1], HAMED A. ELSIMARY[2], MAGDY A. EL-MOURSY[3], HASSAN MOSTAFA[4,5], (Senior Member, IEEE), AND AHMED KHATTAB[5], (Senior Member, IEEE)

[1]Microelectronics Department, Electronics Research Institute (ERI), Cairo 12622, Egypt
[2]Computer Science Department, Cairo Higher Institute for Engineering, Computer Science, and Management, New Cairo 11477, Egypt
[3]Siemens EDA, Wilsonville, OR 97070, USA
[4]Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, Giza 12578, Egypt
[5]Electronics and Electrical Communications Engineering Department, Cairo University, Giza 12613, Egypt

Corresponding author: Ahmed Khattab (akhattab@eng.cu.edu.eg)

ABSTRACT A convolutional neural network (CNN) classifies images with high accuracy. However, CNN operation requires a large number of computations which consume a significant amount of power when implemented on hardware. Precision scaling has been recently used to reduce the hardware requirements and power consumption. In this paper, we present an energy-efficient precision-scaled CNN (EEPS-CNN) architecture. Furthermore, the Field Programmable Gate Array (FPGA) is reconfigured during run time using Dynamic Partial Reconfiguration (DPR). If the battery level decreases, the EEPS-CNN design with the most appropriate power consumption is configured on the FPGA. DPR enables recognition applications to run at a low power budget while sacrificing minor accuracy instead of termination. The proposed architecture is implemented on Xilinx XC7Z020 FPGA and is evaluated on three datasets: MNIST, F-MNIST, and SVHN datasets. The results show a 2.2X, 2.39X, and 2.38X reduction in the energy consumption, respectively, while using only 7 bits to represent all inputs and network parameters. The accuracy of the proposed EEPS-CNN is only 0.53%, 3.67%, and 0.88% less than 32-bit floating-point architectures for MNIST, F-MNIST, and SVHN, respectively. Moreover, the results show up to 92.91X and 4.84X reductions in the power and energy consumption of the proposed EEPS-CNN compared to related designs developed for the MNIST dataset.

INDEX TERMS Approximate computing, convolutional neural network, dynamic partial reconfiguration, energy efficiency, precision scaling.

I. INTRODUCTION

Neuromorphic computing simulates human brain activity using very-large-scale integration (VLSI) systems [1]. Convolutional neural networks (CNNs) were inspired by the human brain for computer vision applications such as image and video classification, object detection, face recognition, image segmentation, image matching, image super-resolution, regression and image correction. Due to the impressive performance of CNNs, they have been recently widely used in various application domains including - but not limited to - Natural Language Processing (NLP) applications, time series prediction, medical signal identification, autonomous driving and security applications [2]. For these numerous applications, the implementation of a low power consumption design is needed. The implementation of a CNN using general-purpose processors consumes a large amount of power because such processors are not optimized. A minor accuracy loss is accepted since CNNs are mostly used in error-reliance applications.

Thus motivated, approximate computing has been recently used to reduce the complexity of CNN implementations.

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria De Munari.

By applying approximate computing techniques, a significant improvement in energy efficiency is achieved with a minor loss in the achieved accuracy. For instance, the authors of [3] used various arithmetic approximate methods and parallel computation to improve the performance of the hardware. Moreover, hybrid high radix encoding multiplier and Winograd convolution were used. Approximate multipliers were exploited to decrease the hardware cost while improving the neural network accuracy in [4]. In this paper, two types of approximate multipliers were used, which were deliberately designed using Cartesian Genetic Programing (CGP). The Mixed National Institute of Standards and Technology (MNIST) and Street View House Numbers (SVHN) datasets were used to evaluate the proposed techniques.

In [5], an approximate floating-point multiplier was introduced using two approximation techniques. The results showed considerable energy reduction with a minor reduction in the classification accuracy when evaluated using the MNIST and the Canadian Institute For Advanced Research (CIFAR-10) [6] datasets. Meanwhile, a small area, low power and high-speed CNN architecture was proposed in [7]. Approximate MAC (Multiply and Accumulate) units for convolution and fully connected layers while using parallel memory access were introduced. A stochastic neural network architecture was introduced while approximating the activation function in [8]. The results showed a reduction in power, energy and area while achieving similar accuracy to the conventional convolutional neural network.

In [9], efficient implementation of an Artificial Neural Network (ANN) was presented using approximate adders and approximate multipliers. This ANN architecture was introduced while using time-multiplexing architecture. In [10], the highest throughput was reached by implementing adaptive switching between shallow and deep networks, and a new CNN architecture was proposed. A novel architecture to implement CNN was proposed in [11]. The proposed CNN architecture was trained on MATLAB for digit recognition for the MNIST dataset.

Several works also tackled the implementation of neural networks on Field Programmable Gate Arrays (FPGAs). The authors of [12] represented the ANN parameters using fixed-point numbers while using chip-memory only for handwritten digit recognition and phoneme recognition. The authors of [13] proposed approximate MAC units to design CNN accelerator using Xilinx XCZU9EG- 2ffvb1156 FPGA chip. This design was used to implement LENET-5 CNN to recognize MNIST dataset using high-level synthesis tool. A new approach for CNN implementation on FPGA was presented for the LENET network using 8 bits fine-tuning to represent the network parameters in [14]. The LENET CNN architecture was dynamically reconfigured to recognize two different datasets in [15]. In [16], a parallel convolutional acceleration unit was introduced. Semi-floating point was used to represent feature maps and weights while using fixed point to perform convolutional layer operations. LENET-5 CNN was implemented on ZCU102 using the proposed design. In [17], the hls4ml library was used to implement a neural network that was trained to recognize the MNIST dataset on FPGA.

In [18], ResNet-18 was implemented on Xilinx XC7VX690T, and 16 bits fixed-point arithmetic was used. In [19], the Alex network was implemented on ZYNQ-702 FPGA, and Vivado 2015.1 tool was used for synthesis. This design was tested with and without pipelining to assess the time and power consumption. In [20], a hardware accelerator design was developed to recognize the MNIST dataset. Python programming language was used to model the provided deep neural network, and the function of Register Transfer Level (RTL) was tested on ModelSim. Finally, the architecture was implemented on Xilinx Zynq ZC-702. In [21], hardware-software co-design was developed for neural network applications on the PYNQ-Z2 board. For this aim, convolutional IP cores were implemented, and they were used as Python overlays. In addition, the convolutional IP core was used to accelerate the recognition of the MNIST dataset.

A new structure for binary convolution was proposed in [22] with the aim of decreasing the consumed power and the hardware resources. In addition, full-BNN (Binary Neural Network) and mixed-precision BNN were proposed. Finally, the MNIST dataset was used to test the two proposed neural networks on the DSP + Xilinx 352T FPGA board. In [23], a new approach for implementing a Fully Connected Deep Neural Network (FC DNN) and convolutional neural network on FPGA was proposed. For the FC DNN, a minimum number of computational units was used, while for the CNN, parallel processing, as well as systolic architecture were exploited. A CNN architecture was trained while using different floating-point formats in [24]. In addition, the MNIST dataset was used to verify the proposed accelerator engine on FPGA. Verilog was used to implement the proposed design in RTL, whilst it was verified by Vivado Simulator. In [25], LENET-5 CNN was implemented on FPGA. Moreover, the CNN was accelerated using the parallelization of the operations. The MNIST and other datasets were used to evaluate the proposed design.

Furthermore, partial configuration was utilized to overcome the FPGA resource limitations. This design was tested on three datasets, namely, CIFAR-10, CIFAR-100 and SVHN. A CNN was implemented on Intel Cyclone 10 FPGA to recognize MNIST dataset's handwritten digits in [26]. Fixed-point representation was used to represent all the network weights and all the intermediate operations. Xilinx XC7A100T FPGA was used to implement a CNN to recognize the MNIST dataset in [27]. The CNN was trained using MATLAB 2018. Multiplication and addition operations were performed using fixed-point representation.

Pynq-Z2 FPGA [28], which contains DPR, was used to implement convolutional layers represented by 8, 16, and 32-bit precision. The DPR feature in ZYNQ 7020 was utilized to realize different CNNs and SNNs (spiking neural networks) on the same FPGA [29]. In [30], the DPR feature in the

ZYNQ 702 board was used to accelerate the cryptocurrency dash mining.

In this paper, we present the energy-efficient precision-scaled CNN architecture (EEPS-CNN) in which the CNN is approximated through precision scaling for all network parameters, which reduces the power consumption for the whole CNN hardware architecture, including memory, multipliers, adders, and registers. This contrasts with the related literature in which precision scaling was used for only one component or one layer of the CNN. In EEPS-CNN, the CNN parameters are approximated by removing the least significant bits from the calculation. The majority of such bits are the bits of the fraction part. Instead of using 16 bits to represent each parameter, a lower number of bits has been used with an insignificant loss in accuracy and a significant saving in power and energy consumption. Furthermore, the proposed approximated CNN is implemented on an FPGA using Dynamic Partial Reconfiguration (DPR) in which different bit-widths are used during the run-time to represent the network parameters to recognize the MNIST, F-MNIST, and SVHN datasets. The proposed convolution neural networks are implemented and tested to evaluate their energy and accuracy performance using fixed-point representation for each parameter.

The remainder of the paper is organized as follows. Section II presents the proposed EEPS-CNN architecture. The details of the hardware architecture developed to implement the proposed EEPS-CNN is discussed in Section III. The experimental results are presented in Section IV. Finally, we conclude the paper in Section V.

II. ENERGY-EFFICIENT PRECISION-SCALED CONVOLUTIONAL NEURAL NETWORK (EEPS-CNN)

In this section, we present the proposed energy-efficient CNN architecture and how precision scaling is used to reduce its energy consumption.

A. BASIC CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

CNN is an ideal candidate for image recognition applications. However, many factors such as the training data, the preprocessing of data and the used optimizer (SGD, Adams) need to be carefully considered in order to achieve a high recognition accuracy. A CNN consists of two stages: a feature extraction stage, and a classification stage. The feature extraction stage consists of alternating convolution (Conv) and pooling layers followed by the classification stage, which consists of a number of fully connected (FC) layers. Convolution layers use a set of filters to extract the features from the input and generate feature maps as output. A nonlinear piecewise activation function is used after each convolution layer. The rectifier linear unit (ReLU) function is typically used in CNN because of its simple computation. The activation function of the non-linear ReLU [31] is given by

$$f(z) = max(0, z). \tag{1}$$

At the pooling layer, the input is divided into rectangular regions. Then, an average or a maximum of each region is generated at the output. The pooling layer is used to down-sample the input representation. This reduces the computational complexity and the memory usage of the network. The pooling layer is described by its kernel size and stride. In fully connected layers, the neurons are fully connected by different weights.

In this paper, we propose the energy-efficient precision-scaled CNN (EEPS-CNN) architecture which is structured as two alternating convolutional and pooling layers, followed by two fully connected layers as shown in Figure 1. The proposed EEPS-CNN architecture is then used to design three CNNs for the three considered datasets resulting in the *(EEPS-CNN-1)* for MNIST dataset,[1] *(EEPS-CNN-2)* for F-MNIST dataset, and *(EEPS-CNN-3)* for SVHN dataset. The MNIST dataset is handwritten digits with a dimension 28 × 28 gray scale [33]. The F-MNIST (Fashion MNIST) is a dataset of Zalando's article images with a dimension 28 × 28 gray scale [34]. The SVHN (Street View House Numbers) dataset is a real-world image dataset that is obtained from house numbers in Google Street View images [35]. The SVHN dataset is a real-world problem of recognizing digits and numbers. SVHN is 32 × 32 red-green-blue (RGB) color images.

The designs of the proposed EEPS-CNN architecture have the first convolutional layer with 2 filters for the MNIST dataset and 4 filters for the F-MNIST and SVHN datasets, with a dimension 3 × 3 and a stride length of one. The input image to the input convolution layer is padded to preserve its spatial size. The second convolutional layer has 4 filters for the MNIST dataset and 8 filters for the F-MNIST and SVHN datasets. Each convolution layer is followed by a ReLU activation function. A 3 × 3 filter dimension is used because a small filter size captures the fine details of the image, while a bigger filter size leaves out small details in the image. The selected 3 × 3 filter dimension needs a small number of multiplications, which reduces the power consumption of the hardware implementation. The pooling layer is MaxPool with a dimension 2 × 2 and a stride length of two. The first fully connected layer has 20 neurons for the MNIST dataset and 256 neurons for the F-MNIST and SVHN datasets and is followed by a ReLU activation function. The second fully connected layer has 10 neurons and is followed by softmax. Softmax is another activation function that is applied to the CNN layer [36]. Softmax converts the output of the last layer of the CNN into a probability distribution.

The network parameters and the number of multiplications for each layer in each EEPS-CNN design are shown in Table 1 for MNIST, F-MNIST, and SVHN, where maps are the number of features extracted at each convolution layer. The number of layers, filters, and neurons is selected after making a

[1]A preliminary design of the architecture developed for the MNIST dataset was presented in [32].

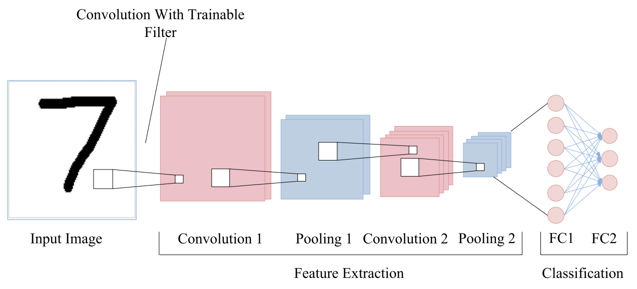


FIGURE 1. Proposed EEPS-CNN architecture.

trade-off between the network accuracy and the number of multiplications.

It is worth noting that reducing the number of multiplications reduces the time needed to recognize the image, and hence, the energy consumption is reduced. Few filters are applied to reduce the number of multiplications. These filters have small dimensions 3×3. In addition, we use a small number of neurons at the fully connected layers.

B. PRECISION SCALED CNN APPROXIMATION

For digital circuits, approximate computing via precision scaling achieves reasonable power savings [37]. Precision scaling allows computing using fewer bits, which reduces the switching activity, and consequently, reduces the dynamic power consumption. Dynamic power is reduced linearly for adders and registers and quadratically for multipliers [38]. To apply approximate computing through precision scaling, the network weights and inputs are quantized and represented with n bits using fixed-point number representation. The same number of bits is used for all CNN parameters, which is denoted by uniform quantization [38].

The CNN parameters and test dataset should be quantized to the hardware's available bitwidth n. The weights of each layer should be examined separately to determine the maximum and minimum values for dividing the available bitwidth n into integer and fractional values, as shown in Figure 2, where m is the number of bits required to represent the integer part using:

$$m = \begin{cases} 0, & \text{if Max} < 1 \text{ \& Min} > \text{-1}. \\ \lceil Max(log_2(Max), log_2(|\ Min\ |)) \rceil, & \text{otherwise} \end{cases} \quad (2)$$

Decreasing the bitwidth n results in reducing the number of bits used to represent the fraction part. Furthermore, the number of bits used to represent the integer part remains constant as long as n is greater than m because the integer part has a greater impact on the CNN accuracy. The inputs and weights for each convolutional (Conv) and fully connected (FC) layer are quantized at n-bit, therefore, the multiplication results from the Conv and FC layers are quantized at $2n$ bits as shown in Figure 3.

The multiplication operations at the convolution layer are shown in Figure 4. In Figure 4, we assume that the input has two channels with a 7×7 dimension, and there is a filter with a 3×3 dimension to illustrate the multiplication and

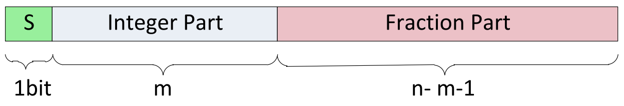


FIGURE 2. Available n bitwidth decomposed into three parts.

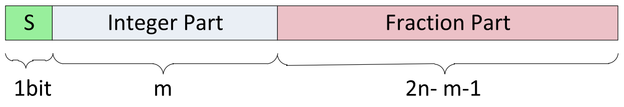
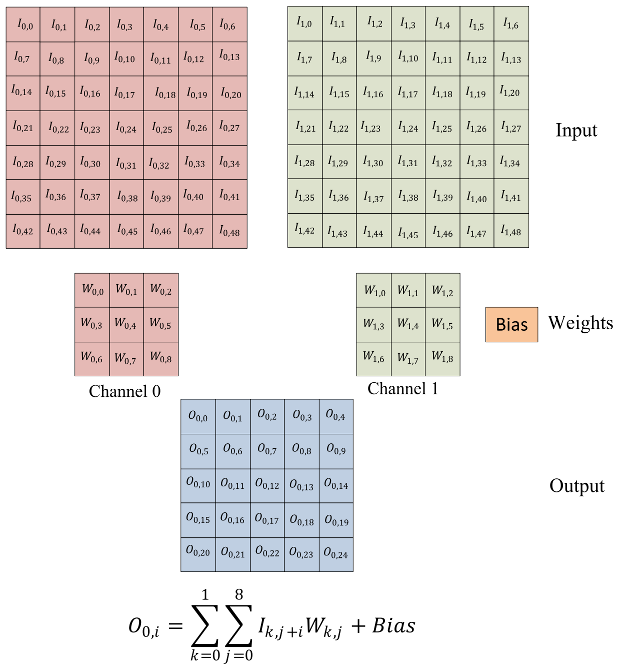


FIGURE 3. Available $2n$ bitwidth decomposed into three parts.



$$O_{0,i} = \sum_{k=0}^{1}\sum_{j=0}^{8} I_{k,j+i}W_{k,j} + Bias$$

FIGURE 4. Example of convolution operation, assuming two channels input with dimension 7×7 input while using one filter. I: input, W: weight, O: output.

accumulation operations. Each layer output is examined to find the maximum and minimum values to decide the number of bits needed to represent the integer part m as shown in Figure 3 using (2) to avoid overflow, which may happen from accumulation or to preserve unneeded bits for the fraction part.

The test function applies repetitive truncation after each multiplication and addition step, as shown in Figure 5. Such repetitive truncation guarantees that the numbers generated from the software test functions can be represented with $2n$ bits. In addition, it guarantees that the fraction part can be represented using $2n - m - 1$ bits. Finally, the last accumulation result, which is represented by $2n$ bits, is truncated to n bits. Truncation is needed to use the same hardware with the same bitwidth for the following layer operations.

III. EEPS-CNN HARDWARE IMPLEMENTATION WITH DYNAMIC PARTIAL RECONFIGURATION

In this section, we present the hardware architecture that we develop to implement the proposed EEPS-CNN. Then,

TABLE 1. Network parameters and number of multiplication for each CNN layer for MNIST, F-MNIST, and SVHN datasets.

Layer Name	MNIST *(EEPS-CNN-1)*		F-MNIST *(EEPS-CNN-2)*		SVHN *(EEPS-CNN-3)*	
	Size	Number of Multiplications	Size	Number of Multiplications	Size	Number of Multiplications
Input Image	28x28	-	28x28	-	32x32	-
Convolution 1	28x28 2 maps	14112	28x28 4 maps	28224	32x32 4 maps	36864
Pooling 1	14x14 2 maps	-	14x14 4 maps	-	16x16 4 maps	-
Convolution 2	12x12 4 maps	5184	12x12 8 maps	10368	14x14 8 maps	14112
Pooling 2	6x6 4 maps	-	6x6 8 maps	-	7x7 8 maps	-
Fully Connected 1	20	2880	256	73728	256	100352
Fully Connected 2-Output	10	200	10	2560	10	2560

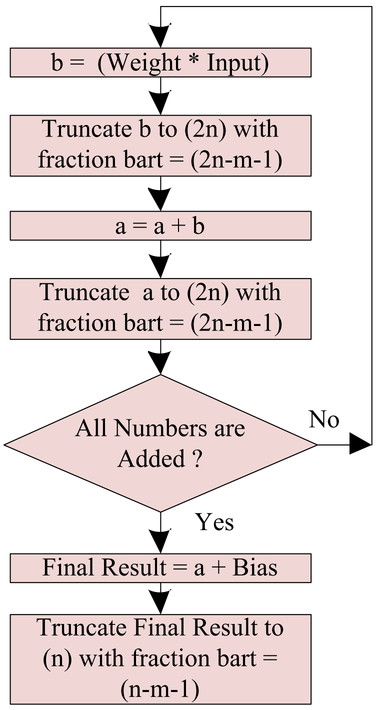

b = (Weight * Input)
Truncate b to (2n) with fraction bart = (2n-m-1)
a = a + b
Truncate a to (2n) with fraction bart = (2n-m-1)
All Numbers are Added ?
No
Yes
Final Result = a + Bias
Truncate Final Result to (n) with fraction bart = (n-m-1)

FIGURE 5. Illustration of the quantization flow.

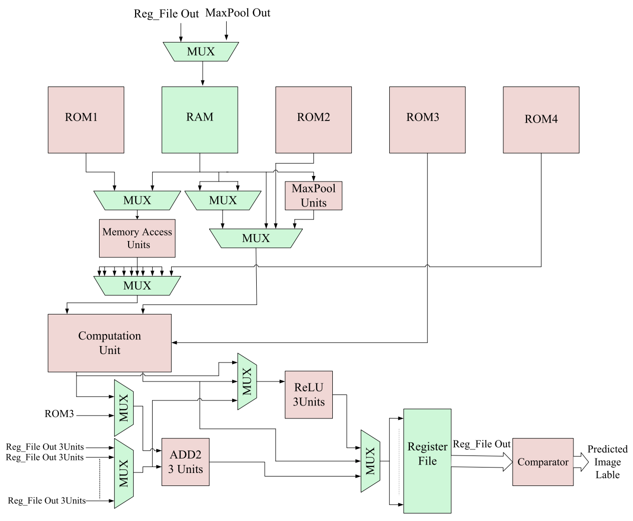


FIGURE 6. EEPS-CNN hardware implementation.

we discuss how we apply dynamic partial reconfiguration to the developed architecture.

A. HARDWARE ARCHITECTURE

The block diagram of the hardware architecture that we develop to recognize the MNIST, F-MNIST, and SVHN datasets is depicted in Figure 6. The developed architecture contains memory units, memory access units (MACs), computation units, ADD2 units, MaxPool units, ReLU units, register files, and one comparator. In our hardware architecture, n-bit fixed-point arithmetic units are used instead of the conventional 32-bit floating-point units, where n takes distinct values such as 16, 12, 10, 8, 7, 6, and 5 bits to reduce the power dissipation [39]. In what follows, we present the details of each component of the developed hardware architecture.

1) MEMORY

The used memory contains four ROMs and one RAM. ROM1 stores the image under test. ROM2 stores the weights of the Conv2 layer. ROM3 stores the biases of all CNN layers. ROM4 stores the weights of the FC1 and FC2 layers. The RAM stores the weights of the Conv1 layer and the intermediate results generated by each layer.

2) COMPUTATION UNIT

The computation and ADD2 units are intended for performing multiplications and additions on the convolutional and fully connected layers. The computation unit contains three processing elements (PEs) as shown in Figure 7. A single PE consists of nine multipliers and four adders as shown in Figure 8. Therefore, three vector multiplication-addition operations are performed at the same time by three processing elements. Figure 8 is a data flow graph (DFG), which demonstrates how the functional units are reused in each cycle. Initially, the operands are multiplied using the nine multipliers. The results are then added using the four adders. After that, some of these adders are reused to finish the addition operations. The final accumulation is achieved using ADD 2-3 Units as shown in Figure 6, which contain other three adders,

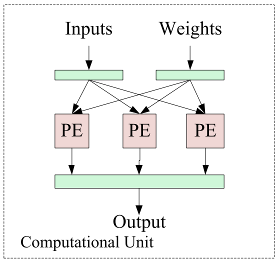


FIGURE 7. Architecture of the computation unit.

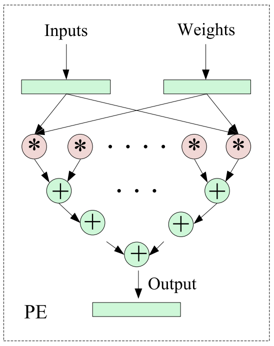


FIGURE 8. Architecture of the processing element.

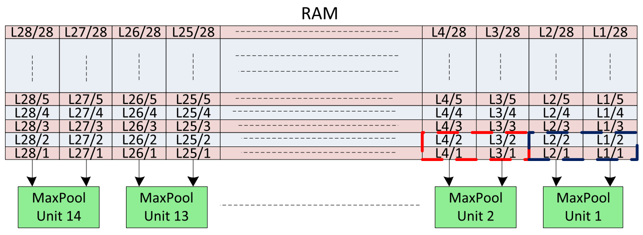


FIGURE 9. Connection between memory and MaxPool unit.

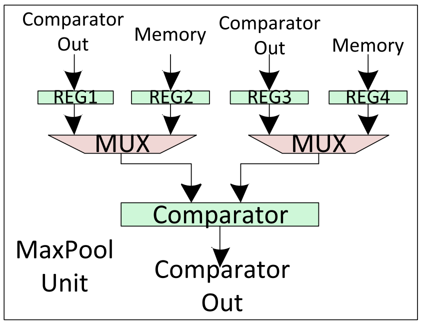


FIGURE 10. Architecture of the MaxPool unit.

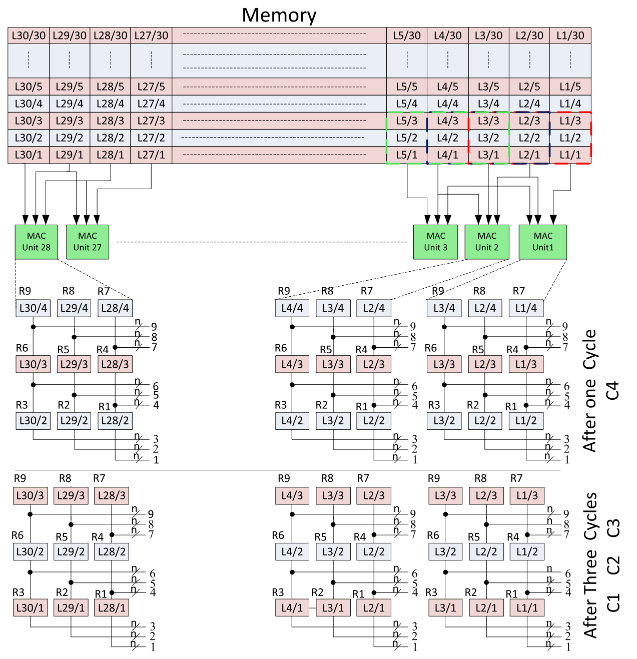


FIGURE 11. Architecture of the memory access units. The dashed squares illustrate how the filters sweep on the input map.

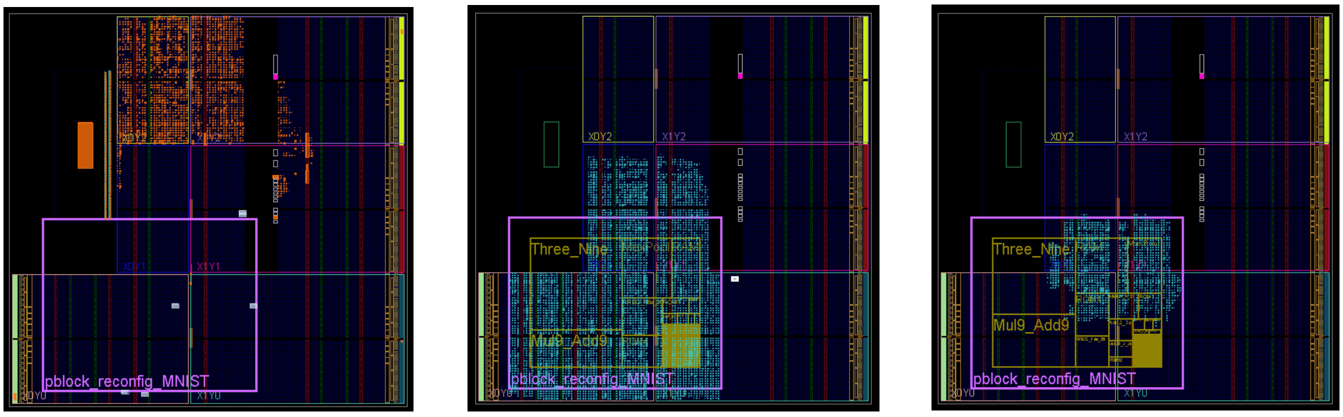
and each adder can add two numbers, Therefore, the results are accumulated in one register at the register file, which is demonstrated in Figure 6. Moreover, these three adders are reused with different data from different resources by using multiplexers.

3) MaxPool UNITS

MaxPool units are used to perform MaxPool layer operations. Fourteen MaxPool units are used for the MNIST and F-MNIST EEPS-CNNs, and sixteen MaxPool units are used for the SVHN EEPS-CNN. Figure 9 illustrates how the MaxPool units are connected to the memory as each MaxPool unit is connected with two columns of RAM. Figure 10 illustrates the architecture of each MaxPool unit which is composed of four registers and one comparator. Initially, the comparator compares two memory locations (*L1/1* and *L2/1* for MaxPool unit 1) and the maximum value is stored at REG1 as shown in Figure 10. Then, a comparison is made between the following two memory locations (*L1/2* and *L2/2* for MaxPool unit 1) and the maximum value is stored at REG3 as shown in Figure 10. Finally, REG1 and REG3 are compared, and the maximum value is stored in the RAM for successive operations of the CNN layers.

4) MEMORY ACCESS (MAC) UNITS

MAC units are used to access each memory row only once to reduce the image recognition time, which in turns reduces the energy consumption required to recognize the image. MAC units are used in the convolutional layer operations. As shown in Figure 11, each MAC unit contains nine *n*-bit registers as the convolution operation is applied to nine values. The nine registers are arranged in clusters of three registers. As shown in Figure 11, after the first three cycles *C1*, *C2*, and *C3*, the memory contents *L1/1*, *L2/1*, *L3/1*, *L1/2*, *L2/2*, *L3/2*, *L1/3*, *L2/3*, and *L3/3* are stored at the first nine registers (MAC unit 1), and the memory contents *L2/1*, *L3/1*, *L4/1*, *L2/2*, *L3/2*, *L4/2*, *L2/3*, *L3/3*, and *L4/3* are stored at the second nine registers (MAC unit 2), and this procedure continues until unit 28. After only one cycle of C4, the memory contents *L1/2*, *L2/2*, *L3/2*, *L1/3*, *L2/3*, *L3/3*,



(a) MNIST static part. (b) MNIST dynamic part (16 bits). (c) MNIST dynamic part (5 bits).

FIGURE 12. Floor planning the MNIST dataset (a) Static part, (b) Dynamic part (16 bits), and (c) Dynamic part (5 bits).

L1/4, *L2/4*, and *L3/4* are stored at the first nine registers (MAC unit 1), and the memory contents *L2/2*, *L3/2*, *L4/2*, *L2/3*, *L3/3*, *L4/3*, *L2/4*, *L3/4*, and *L4/4* are stored at the second nine registers (MAC unit 2), and the same procedure is adopted up to the last MAC unit. Consequently, each successive convolution operation needs only one cycle to access its operands, except for the first convolution operation, which takes three cycles. The output of the Conv1 layer is arranged in a similar way to the arrangement of the input image in the memory to use the MAC units in the following convolution layer operation. For MNIST and F-MNIST datasets, 28 units are needed, but for the SVHN dataset, 32 units are needed.

5) REGISTER FILE, ReLU UNITS, AND COMPARATOR UNIT

Before being stored in the RAM, the multiplication results of the Conv and FC layers are accumulated in the register file. ReLU units are used to perform ReLU activation function operations. The comparator compares the last ten neurons' outputs (i.e., FC2 outputs as shown in Figure 6) and gives the index of the neuron which has the maximum output.

B. DYNAMIC PARTIAL RECONFIGURATION

Dynamic partial reconfiguration (DPR) is a feature available in modern FPGAs to solve the problem of limited hardware resources on FPGAs by allowing the reconfiguration of the programmable logic (PL) on the FPGA during the run time [40]. In DPR, the hardware design is divided into two parts: the static part and the dynamic part. The static part is the common part in all our designs, as it corresponds to the input ports, the output ports and Internal Configuration Access Port (ICAP) which manages the reconfiguration process. Meanwhile, the dynamic part corresponds to the proposed hardware architecture, which include the computational unit and the other needed units that differ according to the target dataset and the number of used bits. For instance, Figure 12 depicts the floor planning of the static part for the MNIST dataset, while Figures 12b and 12c show the floor planning

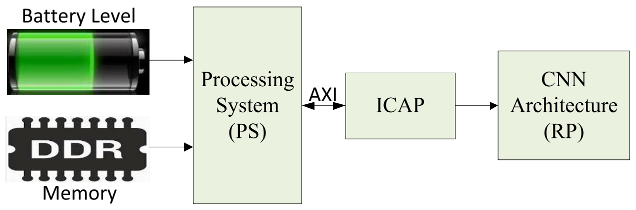


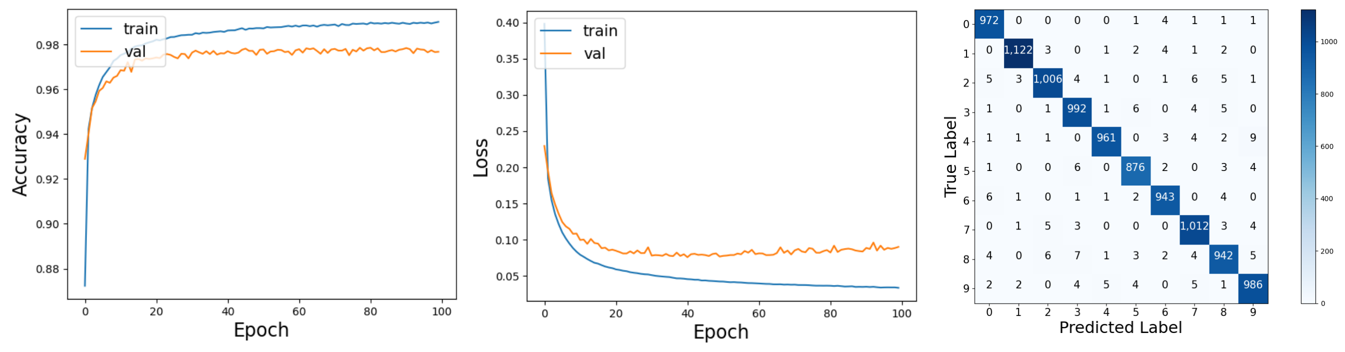
FIGURE 13. DPR system block diagram.

of the dynamic part for the MNIST dataset in the case of 16 bits, and 5 bits, respectively. The static part is configured using a full bit-stream at the boot time, while the dynamic part is configured using partial bit-streams at the run time. The dynamic part consists of one or more reconfigurable partitions (RPs). Each RP is reconfigured with different partial bit-streams without changing the static part. Sharing the same programmable logic between multiple Reconfigurable Modules (RMs) reduces the needed hardware resources. The reconfiguration of the system from an operating design to another needs a reconfiguration time that is a significant factor in DPR. The reconfiguration time is proportional to the size of the partial bit-stream, which is proportional to the size of the reconfigured region.

For the implementation of the proposed EEPS-CNNs, the used FPGA platform is reconfigured with the appropriate power level design during run-time using DPR. Figure 13 shows the block diagram of the developed DPR system. The required partial bit-streams are transferred from DDR to the ICAP by a processing system (PS). Then, the ICAP reconfigures the RPs. According to the available power at the battery, the required partial bit-streams are determined.

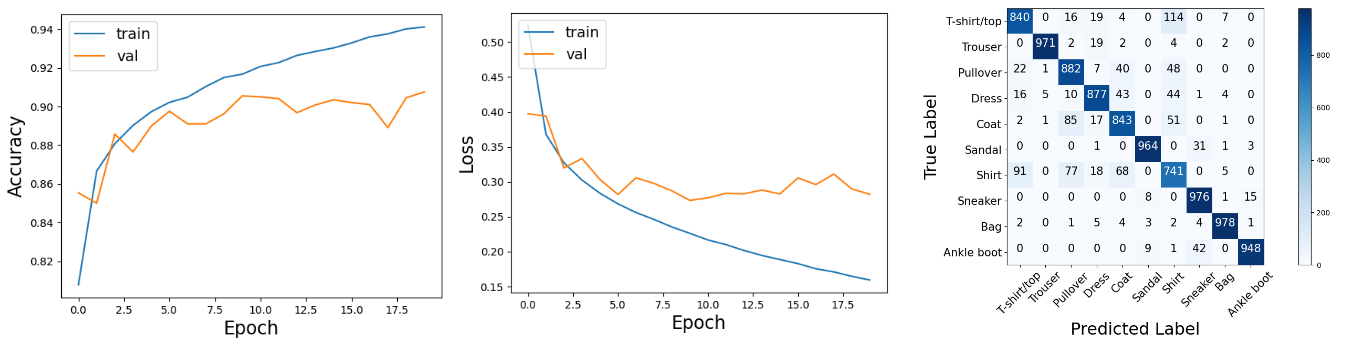
IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we first use Python programming to train and test the performance of the three proposed EEPS-CNN designs. Then, we implement them on an FPGA platform to evaluate their accuracy and hardware characteristics.



(a) MNIST Model Accuracy. (b) MNIST Model Loss. (c) MNIST Confusion Matrix.

FIGURE 14. MNIST dataset (a) Model Accuracy, (b) Model Loss, and (c) Confusion Matrix.



(a) F-MNIST Model Accuracy. (b) F-MNIST Model Loss. (c) F-MNIST Confusion Matrix.

FIGURE 15. F-MNIST dataset (a) Model Accuracy, (b) Model Loss, and (c) Confusion Matrix.

A. EEPS-CNNs TRAINING RESULTS

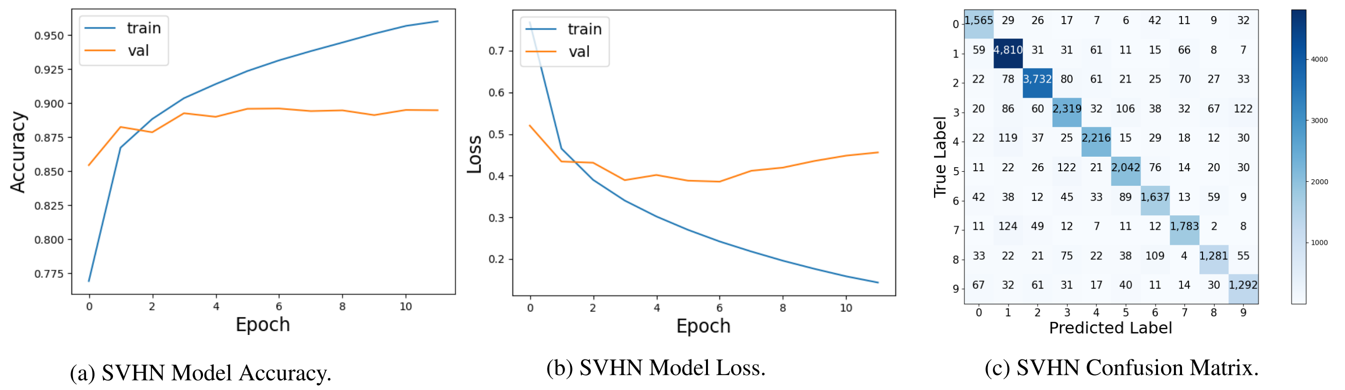
The three designs of the proposed EEPS-CNN architecture are trained using the Python programming language to recognize the three considered datasets: MNIST, F-MNIST, and SVHN. The MNIST and F-MNIST datasets consist of 60,000 samples for training and 10,000 samples for testing. The 60,000 training samples are further divided into 52,200 samples as a training set and 7,800 samples as a validation set. On the other hand, the SVHN dataset consists of 73,257 samples for training and 26,032 samples for testing. The 73,257 training samples are divided into 63,733 samples as a training set and 9,524 samples as a validation set.

For the MNIST and F-MNIST datasets, images are preprocessed through normalization to limit the range of data to the [0-1] range. For the SVHN dataset, the original RGB images are transformed to gray scale using

$$Y = 0.299\, R + 0.587\, G + 0.114B \tag{3}$$

where, R, G, and B are the red, green, and blue components, respectively. Then, the images are preprocessed through standardization by subtracting the mean and dividing the result by the standard deviation.

The designed EEPS-CNNs are trained with a 32 batch size using the *adadelta* optimization algorithm [41]. The test accuracy for the MNIST, Fashoin-MNIST, and SVHN is 98.12%, 90.2%, and 87.11%, respectively. The network accuracy is calculated using inputs and weights represented by 32-bit floating-point number representation on high precision machines using Python programming. As the complexity of the dataset increases, the resulting recognition accuracy decreases. The model accuracy, model loss and confusion matrix for MNIST, Fashion MNIST and SVHN datasets are shown in Figure 14, Figure 15 and Figure 16, respectively. The model accuracy illustrates how the accuracy is improved after each training epoch, while the model loss demonstrates the sum of the errors for each sample in each epoch. For the MNIST dataset, the accuracy improves and settles at a high accuracy and the model loss decreases until it settles at a low value after 18 epochs. However, in the Fashion MNIST dataset, the accuracy improves and settles after only 5 epochs for the validation set, while there is a continuous improvement in the training set accuracy. The same is observed for the decay behavior of the model loss. For the SVHN dataset, the accuracy settles after only 2 epochs for the validation set, whereas there is a continuous increase in the training set accuracy with a similar trend for the decaying model loss. The performance of the CNN is determined using the confusion matrix. The sum of the numbers of the diagonal axis in each



(a) SVHN Model Accuracy. (b) SVHN Model Loss. (c) SVHN Confusion Matrix.

FIGURE 16. **SVHN dataset (a) Model Accuracy, (b) Model Loss, and (c) Confusion Matrix.**

TABLE 2. **Number of bits needed to represent the integer part.**

Layer Name	MNIST (*EEPS-CNN-1*)	F-MNIST (*EEPS-CNN-2*)	SVHN (*EEPS-CNN-3*)
	m-bit	m-bit	m-bit
Conv1	3	2	2
Conv2	4	3	3
FC1	5	4	4
FC2	7	7	6

confusion matrix is equal to the corresponding classification accuracy. The smaller the values of the non-diagonal elements compared to the diagonal elements, the higher the accuracy as the case for the MNIST dataset shown in Figure 14c.

B. EEPS-CNNs TESTING RESULTS

Next, we evaluate the performance of the proposed designs using the test datasets. The network parameters and inputs are quantized and represented by n-bit fixed-point numbers, where $n = 16, 12, 10, 8, 7, 6$, and 5. Also, each layer output for each EEPS-CNN design is examined to decide the number of bits needed to represent the integer part (m) as shown in Table 2.

The resulting accuracy and accuracy loss are listed in Table 3. The accuracy loss is the difference between the accuracy obtained for the 32-bit floating-point operation (which is given as a percentage) and the accuracy obtained for the n-bit fixed-point operation (which is also given as a percentage). Consequently, the accuracy loss is given as a percentage that is the difference between the two percentages calculated as

$$Acc.\,Loss[\%] = Acc.^{Floating\ 32-bit}[\%] - Acc.^{Fixed\ n-bit}[\%] \quad (4)$$

The classification accuracy of the proposed EEPS-CNN design for each dataset, normalized to the fully accurate CNN (32-bit floating point), is shown in Figure 17. These results imply that for MNIST and SVHN EEPS-CNN designs, the accuracy loss is negligible (less than 1%) up to 7 bits, whereas

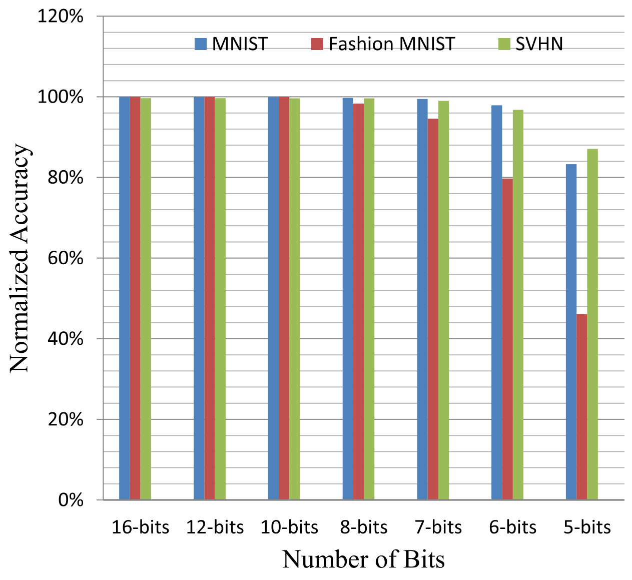


FIGURE 17. **Normalized accuracy of different EEPS-CNN designs.**

for the F-MNIST design the accuracy loss is negligible to 8 bits.

C. HARDWARE IMPLEMENTATION RESULTS

Here, we present the hardware setup used to implement and evaluate the performance of tested EEPS-CNNs. The hardware architecture is modeled by VHDL language, designed using Xilinx Vivado (v.2015.2), and implemented on a Zynq-7000 evaluation board which contains xc7z020clg484-1 FPGA. The proposed hardware architecture is synthesized to recognize the MNIST, F-MNIST, and SVHN datasets. Figure 18a, Figure 18b, and Figure 18c show the floor planning of MNIST, F-MNIST, and SVHN in the case of 16-bits, respectively.

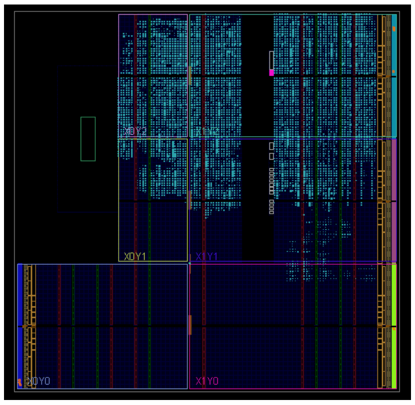
The FPGA resource utilization for the MNIST is shown in Table 4 alongside the reported resource utilization of the ANN [12], LENET CNN [14], LENET-5 CNN [26] and CNN [27] which were all designed to recognize the MNIST

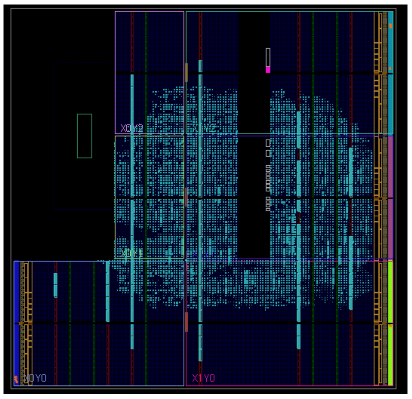
TABLE 3. CNN accuracy and loss for MNIST, F-MNIST, and SVHN datasets.

Number Of Bits	MNIST *(EEPS-CNN-1)*		F-MNIST *(EEPS-CNN-2)*		SVHN *(EEPS-CNN-3)*	
	Accuracy	Accuracy Loss	Accuracy	Accuracy Loss	Accuracy	Accuracy Loss
16-bit	98.12 %	0 %	90.2 %	0 %	86.80 %	0.31 %
12-bit	98.12 %	0 %	90.15 %	0.05 %	86.80 %	0.31 %
10-bit	98.08 %	0.04%	90.07 %	0.13 %	86.785%	0.325 %
8-bit	97.87 %	0.25 %	89.37 %	0.83 %	86.793%	0.31 %
7-bit	97.60 %	0.53 %	86.53 %	3.67 %	86.23%	0.88 %
6-bit	96.07 %	2.09 %	80.21 %	9.99 %	84.3%	2.81 %
5-bit	81.71 %	16.72 %	44.27 %	45.93 %	75.85%	11.26 %

TABLE 4. FPGA resource utilization and Accuracy of EEPS-CNN for MNIST with different bitwidths, ANN [12], LENET CNN [14], LENET-5 CNN [26] and CNN [27].

	Number of Bits	Slice (LUTs)	Slice (Registers)	F7 MUXes	F8 MUXes	DSP	BRAM	Accuracy
Proposed Design	16	18190	8466	1085	166	0	0	98.12 %
	12	12458	6362	741	118	0	0	98.12 %
	10	9921	5350	632	94	0	0	98.08 %
	8	7481	4290	594	71	0	0	97.87 %
	7	6039	3760	501	59	0	0	97.60 %
	6	5360	3230	394	46	0	0	96.07 %
	5	4592	2697	366	-	0	0	81.71 %
ANN [12]	3-bit without DSP	124862	130237	-	-	0	323 (36 Kb)	98.92 %
	3-bit with DSP	121173	130802	-	-	900	323 (36 Kb)	98.92 %
	8-bit	213593	136677	-	-	900	750.5 (36 Kb)	-
LENET CNN [14]	8-bit	52840	81932	-	-	169	148 (18 Kb)	99.107 %
LENET-5 CNN [26]	18-bit	12588	48765	-	-	274	0	97.57 %
CNN [27]	9-bit	15769	106400	-	-	0	73 (36 kb)	90 %


(a) MNIST with 16 bits.


(b) F-MNIST with 16 bits.

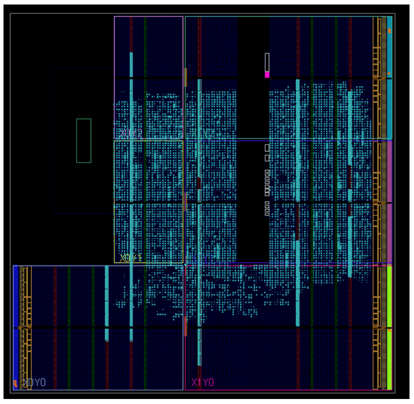

(c) SVHN with 16 bits.

FIGURE 18. Floor planning of EEPS-CNN for (a) MNIST with 16 bits, (b) F-MNIST with 16 bits, and (c) SVHN with 16 bits.

dataset. The hardware design for the ANN [12] is implemented on a Xilinx ZC706 evaluation board which contains XC7Z045 FPGA. The synthesis for LENET CNN [14] is made for the Xilinx Zynq XC7Z020-CLG484 SoC on the ZedBoard development board. Intel Cyclone 10 is used to implement LENET-5 CNN [26] while using fixed-point representation. In [27], the fixed-point representation is used to implement the CNN on the Xilinx XC7A100T FPGA. Table 4 implies that the proposed EEPS-CNN design for the MNIST dataset significantly outperforms the existing designs as the accuracy of the proposed design is less than that of ANN [12] and CNN [14] only by almost 1% and is much higher than the accuracy of LENET-5 CNN [26] and CNN [27]. Moreover, the utilization of the proposed design is orders of magnitude less than those of ANN [12], LANET CNN [14], LENET-5 CNN [26] and CNN [27]. For the F-MNIST, and

TABLE 5. FPGA resource utilization of EEPS-CNN for F-MNIST with different bitwidths.

Number of Bits	Slice (LUTs)	Slice (Registers)	F7 MUXes	F8 MUXes	Block RAM (36 Kb)
16	18672	8239	1268	448	80
12	13447	6191	871	336	60
10	10304	5167	691	280	50
8	7758	4143	554	224	40
7	6328	3631	598	205	35
6	5518	3119	436	168	30
5	4561	2604	434	144	25

TABLE 6. FPGA resource utilization of EEPS-CNN for SVHN with different bitwidths.

Number of Bits	Slice (LUTs)	Slice (Registers)	F7 MUXes	F8 MUXes	Block RAM (36 Kb)
16	19651	9103	1053	512	80
12	13303	6839	820	386	60
10	11122	5707	815	322	50
8	8307	4575	692	256	40
7	6660	4009	606	226	35
6	5978	3443	549	200	30
5	4846	2874	450	168	25

SVHN EEPS-CNN implementation, Table 5 and Table 6 summarize their FPGA resource utilization. For our three EEPS-CNN designs, Tables 4–6 show that when the number of used bits decreases, the needed resources are significantly reduced.

The number of needed cycles to recognize one image for the MNIST dataset is 13715 cycles. With a system clock frequency of 50 MHz, the design recognizes 3645 images per second, and the time needed to recognize one image is $13715\ cycles \times 20\ ns = 0.27\ ms$. The number of needed cycles to recognize one image for the F-MNIST dataset is 97647 cycles, which means that it recognizes 512 images per second, and the time needed to recognize one image is $97647\ cycles \times 20\ ns = 1.95\ ms$. The number of needed cycles to recognize one image for SVHN is 119023 cycles, which are interpreted to a classification throughput of 420 images per second and the time needed to recognize one image is $119023\ cycles \times 20\ ns = 2.38\ ms$. The recognition time and the throughput of the three datasets are summarized in Table 7. As expected, the recognition time increases, and consequently, the throughput decreases as the complexity of the neural network design and the dimension of the input image increase.

Table 8 presents the power consumed by the different components of the three EEPS-CNN implementations for the MNIST, F-MNIST, and SVHN datasets. The energy consumed to recognize one image is calculated as the product of the total consumed power and the image recognition time and also shown in Table 8. The proposed EEPS-CNN architecture uses a fewer number of layers and a fewer number of neurons

TABLE 7. Recognition time and Throughput of three EEPS-CNN implementations.

Dataset	Recognition Time (ms)	Throughput (images/second)
MNIST	0.27	3645.64
F-MNIST	1.95	512.05
SVHN	2.38	420.09

which enable the use of a smaller number of multipliers which saves more energy and power compared to the existing architecture (such as [12]) while achieving reasonable recognition time.

The proposed hardware architecture achieves energy reductions for the 12, 10, 8, 7, 6, and 5 bits cases compared to the 16-bit case. As the number of bits decreases, the switching activity decreases, and hence, the consumed power and energy decreases. More specifically, the energy reductions for the MNIST dataset are 1.32X, 1.57X, 1.83X, 2.2X, 2.54X and 2.75X. Likewise, the energy reductions are 1.38X, 1.65X, 2.07X, 2.39X, 2.76X and 3.25X for the F-MNIST dataset and 1.4X, 1.7X, 2.07X, 2.38X, 2.88X and 3.28X for the SVHN dataset. The tradeoff between the energy and accuracy for the MNIST, F-MNIST, and SVHN EEPS-CNN implementations is depicted by Figure 19. Moreover, the proposed EEPS-CNN for the MNIST dataset in the case of 16-bit achieves 92.91X and 4.84X reductions in the power and energy consumptions compared to [12] as the consumed power and energy are 33 mW and 9.05 μJ, respectively, as shown in Table 8. Whereas, the power consumption of the design presented in [12] is 5 W with static power and 3.066 W without static power. In addition, the energy consumed by the design presented in [12] is 71 μJ with static energy and 43.8 μJ without static energy. Moreover, the power and energy consumption reductions compared to [27] are 29.55X and 4.42X, respectively. Table 9 compares the power, energy per image and the recognition time for the proposed EEPS-CNN design for the MNIST dataset in the case of 16-bit as well as for ANN [12] and CNN [27]. The power reduction of the proposed EEPS-CNN is higher than the energy reduction because the recognition time of the proposed EEPS-CNN design is slightly higher than that of both [12] and [27] (19.18X and 6.69X, respectively).

D. DYNAMIC PARTIAL RECONFIGURATION RESULTS

As mentioned in Section III, DPR is used to reconfigure the FPGA during run-time using the design with the most appropriate power level. The ICAP processor is used to reconfigure the FPGA during the run-time. The throughput of the ICAP processor is 10 MBps. Hence, the reconfiguration time is given by:

$$\text{Reconfiguration Time} = \frac{\text{Partial Bitstream File Size}}{\text{ICAP Throughput}} \tag{5}$$

The partial bitstream file size is equal to 1.27 MB for MNIST implementation and equals to 2.15 MB for both the

TABLE 8. Power and energy consumption per image for the proposed EEPS-CNNs.

Number of Bits	MNIST *(EEPS-CNN-1)*					F-MNIST *(EEPS-CNN-2)*						SVHN *(EEPS-CNN-3)*					
	Power (mW)				Energy	Power (mW)					Energy	Power (mW)					Energy
	Clock	Signal	Logic	Total	(µJ)	Clock	Signal	Logic	BRAM	Total	(µJ)	Clock	Signal	Logic	BRAM	Total	(µJ)
16-bit	14	9	10	33	9.05	14	12	15	50	91	177.72	15	15	20	45	95	226.14
12-bit	12	6	7	25	6.86	12	8	9	37	66	128.89	12	10	12	34	68	161.87
10-bit	11	5	5	21	5.76	11	6	7	31	55	107.41	12	7	9	28	56	133.31
8-bit	10	4	4	18	4.94	10	4	5	25	44	85.93	10	6	7	23	46	109.50
7-bit	9	3	3	15	4.11	9	4	3	22	38	74.21	10	5	5	20	40	95.22
6-bit	8	3	2	13	3.57	8	3	3	19	33	64.45	8	4	4	17	33	78.56
5-bit	8	2	2	12	3.29	7	3	2	16	28	54.68	8	4	3	14	29	69.03

TABLE 9. Power, energy per image and the recognition time of EEPS-CNN for MNIST in case of 16-bit as well as for ANN [12] and CNN [27]. A fractional reduction implies an increment rather than a decrement in value.

	EEPS-CNN (16-bit)	ANN [12]	Reduction	CNN [27]	Reduction
Power (Watt)	0.033	3.066	92.91X	0.975	29.55X
Energy/Image (µJ)	9.05	43.8	4.84X	39.98	4.42X
Recognition Time (ms)	0.2743	0.0143	$\frac{1}{19.18}X$	0.041	$\frac{1}{6.69}X$

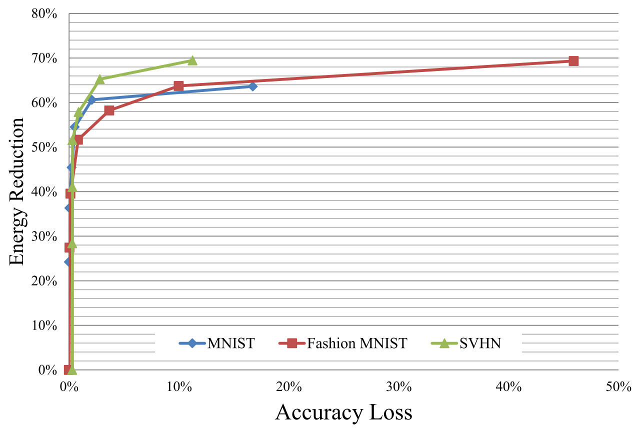


FIGURE 19. Accuracy loss vs. energy reduction with approximated CNN.

F-MNIST and SVHN implementations. According to (5), the reconfiguration times of the MNIST, F-MNIST and SVHN EEPS-CNN implementations are only 127 ms, 215 ms and 215 ms, respectively.

V. CONCLUSION

In this paper, an efficient architecture to reduce the CNN energy consumption has been proposed. The consumed power has been reduced through precision scaling. Three energy-efficient precision scaled CNNs have been proposed for the MNIST, F-MNIST, and SVHN datasets. The proposed EEPS-CNN designs have been implemented using Xilinx Vivado (v.2015.2) and deployed on FPGA. Our experiments have demonstrated 2.2X, 2.39X, and 2.38X reduction in the energy consumption with a maximum of 0.53%, 3.67%, and 0.88% loss in CNN accuracy for MNIST, F-MNIST, and SVHN designs, respectively, while using 7-bit to represent all network parameters, as compared to 16-bit. The experiments have also shown 92.91X and 4.84X reduction in the power and energy consumptions while having less than a 1% loss in accuracy compared to existing hardware implementations. We have further exploited DPR to reconfigure the FPGA with the design with the most appropriate power level during the run time if the battery level is decreased. Such DPR has ensured continuity instead of termination at the expense of image recognition accuracy.

Finally, it is worth mentioning that the uniform quantization method optimized for the widely used MNIST, F-MNIST and SVHN datasets in this paper can be applied for other CNN architectures in which the difference in the sensitivity of the CNN layer is not significant. However, CNN architectures in which different layers have different sensitivities, non-uniform quantization might be needed. Our future work will investigate the generalization of quantization for other networks and datasets while relating the CNN layers' sensitivities to the used quantization approach.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of the Cloud Computing Center of Excellence at the Electronics Research Institute (ERI) in Egypt and for providing access the center's Cloud and High Performance Computing facilities to conduct the research presented in the paper.

REFERENCES

[1] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu, and Q. Qiu, "A survey on neuromorphic computing: Models and hardware," *IEEE Circuits Syst. Mag.*, vol. 22, no. 2, pp. 6–35, 2nd Quart., 2022.

[2] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 10, 2021, doi: 10.1109/TNNLS.2021.3084827.

[3] G. Lentaris, G. Chatzitsompanis, V. Leon, K. Pekmestzi, and D. Soudris, "Combining arithmetic approximation techniques for improved CNN circuit design," in *Proc. 27th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2020, pp. 1–4.

[4] 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 Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Oct. 2020.

[5] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Pekmestzi, ''Improving power of DSP and CNN hardware accelerators using approximate floating-point multipliers,'' *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–21, Sep. 2021.

[6] *The CIFAR-10 Dataset*. Accessed: Feb. 2022. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[7] M. E. Elbtity, H.-W. Son, D.-Y. Lee, and H. Kim, ''High speed, approximate arithmetic based convolutional neural network accelerator,'' in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2020, pp. 71–72.

[8] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, ''Neural network classifiers using a hardware-based approximate activation function with a hybrid stochastic multiplier,'' *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 1, pp. 1–21, Jan. 2019.

[9] M. E. Nojehdeh, L. Aksoy, and M. Altun, ''Efficient hardware implementation of artificial neural networks using approximate multiply-accumulate blocks,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 96–101.

[10] M. Farhadi, M. Ghasemi, and Y. Yang, ''A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on FPGA,'' in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–7.

[11] P. Kumar, H. Yingge, I. Ali, Y.-G. Pu, K.-C. Hwang, Y. Yang, Y.-J. Jung, H.-K. Huh, S.-K. Kim, J.-M. Yoo, and K.-Y. Lee, ''A configurable and fully synthesizable RTL-based convolutional neural network for biosensor applications,'' *Sensors*, vol. 22, no. 7, p. 2459, Mar. 2022.

[12] J. Park and W. Sung, ''FPGA based implementation of deep neural networks using on-chip memory only,'' in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 1011–1015.

[13] M. Cho and Y. Kim, ''FPGA-based convolutional neural network accelerator with resource-optimized approximate multiply-accumulate unit,'' *Electronics*, vol. 10, no. 22, p. 2859, Nov. 2021.

[14] A. Hadnagy, B. Feher, and T. Kovacshazy, ''Efficient implementation of convolutional neural networks on FPGA,'' in *Proc. 19th Int. Carpathian Control Conf. (ICCC)*, May 2018, pp. 359–364.

[15] H. Irmak, D. Ziener, and N. Alachiotis, ''Increasing flexibility of FPGA-based CNN accelerators with dynamic partial reconfiguration,'' in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 306–311.

[16] Y. Shi, T. Gan, S. Jiang, Y. Shi, T. Gan, and S. Jiang, ''Design of parallel acceleration method of convolutional neural network based on FPGA,'' in *Proc. IEEE 5th Int. Conf. Cloud Comput. Big Data Analytics (ICCCBDA)*, Apr. 2020, pp. 133–137.

[17] F. Alfonsi, A. Gabrielli, and E. Ronchieri, ''Neural nets on FPGA a machine vision algorithm applied on MNIST dataset using Hls4ml library,'' in *Proc. Int. Conf. Comput. Sci. Appl.* Cham, Switzerland: Springer, 2020, pp. 597–605.

[18] S. Kala and S. Nalesh, ''Efficient CNN accelerator on FPGA,'' *IETE J. Res.*, vol. 66, no. 6, pp. 733–740, Nov. 2020.

[19] S. Hareth, H. Mostafa, and K. A. Shehata, ''Low power CNN hardware FPGA implementation,'' in *Proc. 31st Int. Conf. Microelectron. (ICM)*, Dec. 2019, pp. 162–165.

[20] J. Kwon and S. Kim, ''Design of a low-area digit recognition accelerator using MNIST database,'' *Int. J. Informat. Visualizat.*, vol. 6, no. 1, pp. 53–59, 2022.

[21] T. V. Huynh, ''FPGA-based acceleration for convolutional neural networks on PYNQ-Z2,'' *Int. J. Comput. Digit. Syst.*, vol. 11, no. 1, pp. 441–450, 2022.

[22] L. Zhang, X. Tang, X. Hu, T. Zhou, and Y. Peng, ''FPGA-based BNN architecture in time domain with low storage and power consumption,'' *Electronics*, vol. 11, no. 9, p. 1421, Apr. 2022.

[23] A. K. Mukhopadhyay, S. Majumder, and I. Chakrabarti, ''Systematic realization of a fully connected deep and convolutional neural network architecture on a field programmable gate array,'' *Comput. Electr. Eng.*, vol. 97, Jan. 2022, Art. no. 107628.

[24] M. Junaid, S. Arslan, T. Lee, and H. Kim, ''Optimal architecture of floating-point arithmetic for neural network training processors,'' *Sensors*, vol. 22, no. 3, p. 1230, Feb. 2022.

[25] J. N. Pisharody, K. B. Pranav, M. Ranjitha, and B. Rajeshwari, ''FPGA implementation and acceleration of convolutional neural networks,'' in *Proc. 6th Int. Conf. Converg. Technol. (I2CT)*, Apr. 2021, pp. 1–4.

[26] R. Xiao, J. Shi, and C. Zhang, ''FPGA implementation of CNN for handwritten digit recognition,'' in *Proc. IEEE 4th Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Jun. 2020, pp. 1128–1133.

[27] D. Giardino, M. Matta, F. Silvestri, S. Spanò, and V. Trobiani, ''FPGA implementation of hand-written number recognition based on CNN,'' *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 9, no. 1, pp. 167–171, 2019.

[28] D. Cain, O. Eldash, K. Khalil, and M. Bayoumi, ''Convolution processing unit featuring adaptive precision using dynamic reconfiguration,'' in *Proc. IEEE 7th World Forum Internet Things (WF-IoT)*, Jun. 2021, pp. 592–597.

[29] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, ''A dynamic reconfigurable architecture for hybrid spiking and convolutional FPGA-based neural network designs,'' *J. Low Power Electron. Appl.*, vol. 11, no. 3, p. 32, Aug. 2021.

[30] M. H. Abdulmonem, J. Essameddeen, M. H. Zakhari, S. Hanafi, and H. Mostafa, ''Hardware acceleration of dash mining using dynamic partial reconfiguration on the ZYNQ board,'' in *Proc. 32nd Int. Conf. Microelectron. (ICM)*, Dec. 2020, pp. 1–4.

[31] K. Hara, D. Saito, and H. Shouno, ''Analysis of function of rectified linear unit used in deep learning,'' in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[32] E. Youssef, H. A. Elsemary, M. A. El-Moursy, A. Khattab, and H. Mostafa, ''Energy adaptive convolution neural network using dynamic partial reconfiguration,'' in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2020, pp. 325–328.

[33] Y. LeCun, C. Cortes, and C. J. Burges. *The MNIST Database of Handwritten Digits*. Accessed: Feb. 2022. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[34] H. Xiao, K. Rasul, and R. Vollgraf, ''Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,'' 2017, *arXiv:1708.07747*.

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. *The Street View House Numbers (SVHN) Dataset*. Accessed: Feb. 2022. [Online]. Available: http://ufldl.stanford.edu/housenumbers/

[36] R. A. Dunne and N. A. Campbell, ''On the pairing of the softmax activation and cross–entropy penalty functions and the derivation of the softmax activation function,'' in *Proc. 8th Aust. Conf. Neural Netw.*, vol. 181. Melbourne, VI, Australia, Jun. 1997, p. 185.

[37] B. Moons and M. Verhelst, ''DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing,'' in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2015, pp. 237–242.

[38] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, ''Energy-efficient ConvNets through approximate computing,'' in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2016, pp. 1–8.

[39] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, ''Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,'' *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014.

[40] XILINX. (2017). *UG909: Vivado Design Suite User Guide—Partial Re-Configuration (v2017.1)*. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf

[41] Keras. *Keras Optimizers*. Accessed: Feb. 2022. [Online]. Available: https://keras.io/optimizers/

EMAN YOUSSEF received the B.Sc. degree in electronics and electrical communication engineering from Al-Azhar University, Egypt, in 2014, and the M.Sc. degree in electronics and electrical communication engineering from Cairo University, Egypt, in 2021. She worked at the Integrated Technical Education Cluster, from 2014 to 2016. She is currently working as a Research Assistant at the Electronic Research Institute. Her research interest includes the design of FPGA/ASIC.

HAMED A. ELSIMARY received the Ph.D. degree in electronics and electrical communications from Cairo University, Egypt, in 1993. He has a Fellowship to perform the Ph.D. research at the Rochester Institute of Technology, USA. He is currently a Professor in the Computer Science Department at Cairo Higher Institute for Engineering, Computer Science, and Management. He has conducted Postdoctoral Research at Ohio State University. He has been a member of the Computer Engineering Department, Prince Sattam Bin Abdulaziz University, Saudi Arabia, and a member of the VLSI Department, Electronics Research Institute, Cairo, Egypt. His research interests include computer architecture and VLSI circuit design.

MAGDY A. EL-MOURSY received the B.S. degree (Hons.) in electronics and communications engineering and the master's degree in computer networks from Cairo University, Cairo, Egypt, in 1996 and 2000, respectively, and the master's and Ph.D. degrees in electrical engineering in the area of high-performance VLSI/IC design from the University of Rochester, Rochester, NY, USA, in 2002 and 2004, respectively. In Summer of 2003, he was with STMicroelectronics, Advanced System Technology, San Diego, CA, USA. From September 2004 to September 2006, he was a Senior Design Engineer at Portland Technology Development, Intel Corporation, Hillsboro, OR, USA. From September 2006 to February 2008, he was an Assistant Professor at the Information Engineering and Technology Department, German University in Cairo (GUC), Cairo. From 2014 to 2019, he was an Associate Professor at the Microelectronics Department, Electronics Research Institute, Cairo. He is currently a Senior Engineering Manager at the Integrated Circuits Verification Systems Division, Siemens EDA. He is the author of around 100 papers, six book chapters, and five books in the fields of high speed and low power CMOS design techniques and NoC/SoC and embedded systems. His research interests include SW/HW co-design, embedded systems, networks-on-chip/system-on-chip, interconnect design and related circuit level issues in high performance VLSI circuits, clock distribution network design, digital ASIC circuit design, VLSI/SoC/NoC design and validation/verification, circuit verification and testing, and low power design. He is a member of the IEEE VLSI Systems and Applications Technical Committee. He is an Associate Editor with the Editorial Board of Elsevier *Microelectronics* Journal, *Journal of Circuits, Systems, and Computers*, and *International Journal of Circuits and Architecture Design* and a Technical Program Committee of many IEEE Conferences such as ISCAS, ICAINA, PacRim CCCSP, ISESD, SIECPC, and IDT.

HASSAN MOSTAFA (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in electronics engineering from Cairo University, Cairo, Egypt, in 2001 and 2005, respectively, and the Ph.D. degree in electrical and computer engineering from the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2011. He was a NSERC Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON. He was a Postdoctoral Researcher in collaboration with the Fujitsu Research Laboratories in Japan and USA, with a focus on the design of the next-generation FPGA. He is currently an Associate Professor with the Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, Giza, Egypt, on leave from the Department of Electronics and Electrical Communications, Cairo University. He has authored/coauthored more than 170 papers in international journals and conferences and five published books. His research interests include neuromorphic computing, the IoT hardware security, software-defined radio, reconfigurable low-power systems, analog-to-digital converters, low-power circuits, subthreshold logic, variation-tolerant design, soft error-tolerant design, statistical design methodologies, next-generation FPGA, spintronics, memristors, energy harvesting, MEMS/NEMS, power management, and optoelectronics. He has been a member of the IEEE Technical Committee of VLSI Systems and Applications, since 2017. He was a recipient of the University of Waterloo SandFord Fleming TA Excellence Award, in 2008; the Ontario Graduate Scholarship, in 2009; the Waterloo Institute of NanoTechnology Nanofellowship Research Excellence Award, in 2010; the Natural Sciences and Engineering Research Council of Canada Prestigious Postdoctoral Fellowship, in 2011; and the University of Toronto Research Associate Scholarship, in 2012.

AHMED KHATTAB (Senior Member, IEEE) was born in Cairo, Egypt, in 1980. He received the B.Sc. (Hons.) and M.Sc. degrees in electrical engineering from Cairo University, Cairo, Egypt, in 2002 and 2004, respectively, the M.E.E. degree from Rice University, in 2009, and the Ph.D. degree in computer engineering from the Center for Advanced Computer Studies (CACS), University of Louisiana, Lafayette, in 2011. He is currently a Professor with the Electronics and Electrical Communications Engineering Department, Cairo University, where he joined in 2012, as an Assistant Professor. He has authored/coauthored three books, four book chapters, over 100 journal and conference publications, and a U.S. patent. His current research interests include wireless networking including the Internet of Things (IoT), wireless sensor networks, vehicular networks, cognitive radio networks, and security and machine learning. He is a recipient of the Egypt State First Class Medallion of Excellence, in 2019; the Egypt State Encouragement Award for Engineering Sciences, in 2017; and the Cairo University Excellence Award in Advanced Technology Sciences, in 2020. He has also won several awards from different IEEE conferences and societies.