Secure Energy-Efficient Implementation of CNN on FPGAs for Accuracy Dependent Real Time Task Processing

Krishnendu Guha
School of Computer Science and Information Technology
University College Cork
Ireland
Email: kguha@ucc.ie

Amlan Chakrabarti
A. K. Choudhury School of Information Technology
University of Calcutta
India
Email: acakcs@caluniv.ac.in

***Abstract*—A key objective of the fourth industrial revolution or Industry 4.0 is to use processing resources that are reconfigurable and flexible, having additional capability to run smart real time intelligent applications in short time. For this, designers deploy reconfigurable hardware or field programmable gate arrays (FPGAs) in several critical infrastructures, along with cloud and edge platforms. Among the commonly used artificial neural networks, convolutional neural networks (CNNs) is one of the most widely used for various smart applications. Existing strategies of CNN implementation on FPGAs incur significant resources and power, and hence, are not energy efficient. These are even prone to side channel attacks. Moreover, they are associated with prunning and hence, do not generate accurate results, which are important for some real time applications. In our proposed methodology, we pre-compute various operations, mainly operations associated with secret information, which in the present case are weights and bias of the CNN for a particular application and store in available embedded memory blocks (EMBs) of an FPGA. On demand, these pre-computed results are accessed for real time operations. Via this methodology, we eradicate the side channel attacks that steals the weights and biases, obtain low resource utilization, low latency, low power and better energy efficiency, without any loss of accuracy. Such a mechanism is particularly suitable for high accuracy real time smart intelligent applications.**

***Index Terms*—CNN, FPGA, Security, Energy Efficiency, Accuracy, Real Time Tasks;**

I. Introduction

The recent era has witnessed the advent of Industry 4.0, where high speed, accurate and intelligent processing of applications are expected throughout the cloud to things continuum. An additional demand is flexibility of the processing resources. For this, designers deploy reconfigurable hardware or field programmable gate arrays (FPGAs) not only in the end user embedded platforms, but also in the cloud and edge platforms [1]. To deliver smart intelligent solutions in real time, application specific neural networks needs to be created and processed on the embedded devices. However, the bottleneck is the huge resource and power consumption for implementing such neural applications on FPGAs [2], [3].

Present day FPGAs are associated with several embedded memory blocks (EMBs) to facilitate high speed access of data. EMBs are used for different purposes like storage of inputs or intermediate data. However, most parts of the EMBs remain unused for applications that are associated with intensive computations [4]. These unused EMBs can be used to map complex functions and datapaths, resulting in low power consumption and low resource utilization [5]. This will make the systems energy efficient in nature. Such a strategy can be used effectively to map complex neural architectures like convolutional neural networks (CNNs) on FPGAs.

A CNN implementation on FPGAs is associated with significant resource utilization and power consumption. Though these are suitable for cloud environments but using these for resource constrained and low energy budget embedded platforms is an issue. Previous works adhered to pruning strategies to reduce latency and power [6], [7], [2], [3]. Though prunning is an effective technique for fitting CNNs on embedded platforms, but these reduce accuracy and are not suitable for accuracy dependent or hard real time systems.

Security of neural architectures is of significant concern in recent times [8]. Generating a custom neural architecture for a specific application is costly. This is associated with significant amount of training and refinement. Weights and biases are the key, which the adversary wants to steal. An effective way of stealing is via side channel analysis [8], [9]. Hence, it is imperative to generate a mechanism that can prevent such types of attacks.

In this work, we propose a look up in memory computation strategy for energy efficient implementation of computation intensive applications on FPGAs. Additionally, we depict the implementation of CNN on FPGA via this mechanism. For this, we initially determine the available memory spaces and divide them into several memory blocks that can be loaded with pre-computed data. Then for a particular application, which in the present case is CNN, we develop a data flow graph (DFG) to determine the computational nodes and their dependency. Then we fuse the nodes if possible and arrange them in order of their power consumption. Based on the availability of available memory, we map the nodes into a single or a group of memory blocks. We perform this for LeNet 5 CNN model and compare with the state-of-the-art models.

Our proposed mechanism achieves security, result accuracy, low resource consumption, low power consumption and high energy efficiency as obtained in experimental results.

The highlights of this work can be summarized as:
(i) Pre-computation runtime look-up memory access strategy for CNN implementation on reconfigurable hardware.
(ii) Achieving reduction in resource utilization, latency and power than existing approaches that maps CNN on FPGAs.
(iv) Achieving high accuracy without pruning.
(v) Analyzing variation of throughput and power with respect to availability of EMBs, for LeNet 5 CNN model.

The paper is organized as follows. Background is discussed in Section II. Section III deals with threat and other problems. Our proposed mechanism is presented in Section IV. Section V depicts how our proposed mechanism can ensure security and overcome other problems. Experimentation and results is dealt in Section VI and the paper concludes in Section VII.

II. Background and Motivation

A. CNN Fundamentals

CNNs are mostly used in image classification, computer vision and a wide variety of applications [3]. A CNN has several layers like convolution, activation, pooling, etc. However, the structure, weights and biases vary based on the targeted application. The present work considers a CNN model (LeNet5), as depicted in Fig. 1(a). Corresponding implementation of this architecture on FPGA ZynQ XCZU9EG is shown in Fig. 1(b). The FPGA comprises of a processor subsystem (PS) and a programmable logic (PL) component. The PS comprises of a quad core ARM Cortex A53 that is connected to 512 MB 16 bit DDR4, which the PL can also access. PL is associated with a controller that is responsible for controlling the flow of various operations of CNN layers like convolution, pooling, fully connected, etc., while PS is responsible for operation related to the Softmax layer.

After system initialization, the model parameters and datasets are stored in DDR4. PL reads corresponding data from DDR4 to the on-chip memory or BRAM via AXI4 interface. The control module transfers the feature map to the controller that involves various operations. After all the operations of each layer is complete, the result is transmitted to the PS end via the AXI4 interface, where Softmax layer calculation is performed to recognize a character.

B. Related Works for CNN Implementation on FPGAs

1) Optimization based Models: The main issue for implementation of CNN are highly computational intensive operations. A loop unrolling strategy for CNN accelerator designs is shown in [10]. Authors in [11] analyzed the cost of data sharing relation between several applications to infer the cost of unrolling. However, they do not consider feature map and kernel level parallelization, which can further reduce data movement between memory hierarchies and reduce energy cost. Authors in [12] adopt a mesh grid style structure to facilitate parallelization on feature map level. Authors in [13] proposed a 2-D PE array style design to optimize CNN. All

TABLE I: Comparison of Related Works of CNN on FPGA

	Optimization based Models [6], [7], [2], [3]	Security based Models [8], [9]	Proposed
Power Reduction*	Yes	No	Yes
Work Time Reduction*	Yes	No	Yes
Resource Reduction*	Yes	No	Yes
Unused EMB Utilization	No	No	Yes
Result Accuracy	No	Yes	Yes
Security	No	Yes	Yes

*Detailed report on power, latency is given in Table 2.

the designs above achieve intra layer parallelization. Authors in [6], [7], [2] depict optimization and pruning to reduce resource and power consumption. These degrades accuracy, which are not favourable for real-time applications.

2) Security based Models: However, these are susceptible to stealing of secret weights and biases. Adversaries may exploit various side channel analysis like power analysis, timing analysis, number of memory read/ writes to duplicate a neural architecture [8], [9]. Authors in [14] obfuscate the base hardware to alter the runtime traces to prevent the attack. Authors in [15] insert noise to execution traces to eradicate the problem. Shuffling memory read/writes was used in [16]. But authors in [17] develop a neuro unlock mechanism that learns the obfuscation procedures and automatically revert it to understand the neural architecture. Using genetic algorithm to mitigate timing differences was proposed in [18]. However, these are associated with security overheads and do not exhibit optimization of the original design.

Table I presents a comparative study of related works.

C. Handling Computation Intensive Operations on Hardware

Several operations like matrix multiplication and functions like sine, cosine, tanh, sigmoid, exponentiation, logarithm are associated with complex computational operations, which are resource and power consuming, when implemented on hardware platforms like FPGAs [5]. In general, coordinate rotation digital computer (CORDIC) strategy [19] or Taylor series expansion [20] mechanism is utilized to map such transcendental functions for FPGAs. But implementing these strategies are not energy efficient, as they also involve high power and resource utilization. A look up table-based approach for memory based computing was proposed in [21]. In this mechanism, a 2D memory array was used to store all outputs for a functional operation. However, such a technique is not favorable when the memory size grows exponentially, with increase in input operand bit width. To prevent this, a decomposition technique with multiple memory access can be used [22]. Here, a table of size 2^{a+b} is reduced to two tables of sizes 2^a and 2^b. But the work was limited to application specific integrated circuit (ASIC) platforms and did not consider FPGAs. Moreover, mapping complex CNN functionalities and analyzing energy efficiency of such intelligent architectures was also not focused in these works. Even how security can be ensured via this methodology is never focussed.

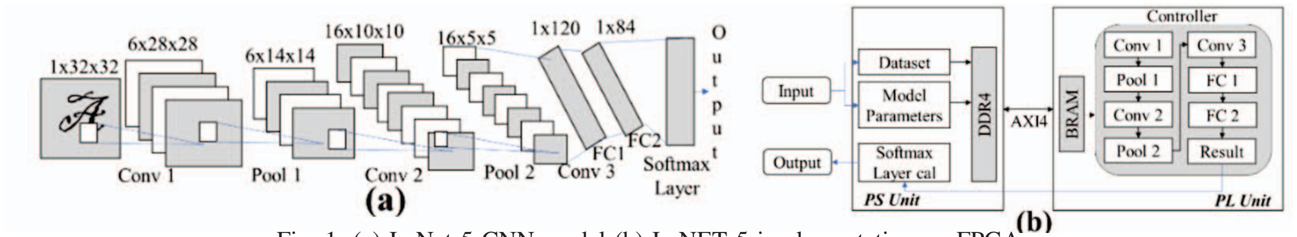

Fig. 1: (a) LeNet 5 CNN model (b) LeNET 5 implementation on FPGA

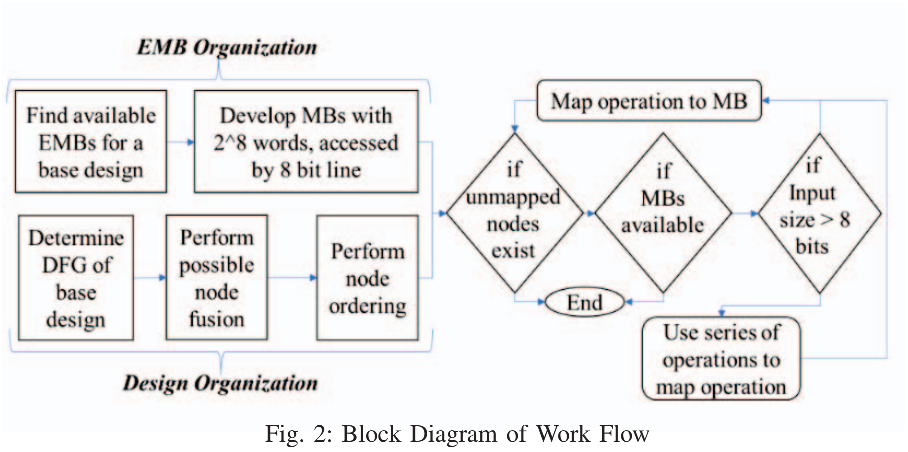
EMB Organization
Find available EMBs for a base design
Develop MBs with 2^8 words, accessed by 8 bit line
Determine DFG of base design
Perform possible node fusion
Perform node ordering
Design Organization
Map operation to MB
if unmapped nodes exist
if MBs available
if Input size > 8 bits
End
Use series of operations to map operation

Fig. 2: Block Diagram of Work Flow

D. Motivation

In general, the bottleneck for efficient performance is the matrix vector multiplications and functions like tanh of the CNN model. Alongside, the loss of weights and biases during transfer from the memory or via side channel analysis. Hence, if we can pre-compute these operations and store them in the EMBs of the FPGA, then the results can be accessed at runtime and no weights and bias needs to be stored separately, that can be stolen by the adversaries. As each operation is not needed to be carried out, the latency will reduce, along with power and make the system more energy efficient. Moreover, result accuracy can be achieved as there is no prunning or optimization of the design.

III. Threat Model and Other Problems

A. Leakage of Confidential Information

Adveraries may get acquinted to the weights and biases that are stored in the memory. This can take place via two ways:

(i) During transit from memory into processing elements.

(ii) Use of side channel parameters like power, timing, number of memory reads and writes to get acquinted with the confidential information of the neural network.

B. Resource, Power and Accuracy

As existing strategies utilize pruning and optimization techniques to reduce the resource and power consumption, they do not provide accurate results.

C. Issue of Timing

As complex operations like matrix multiplication and tanh are associated, they consume a lot of timing. This reduces the amount of slack, which ultimately reduces the throughput.

IV. Proposed Mechanism

The block diagram of the workflow is depicted in Fig. 2.

A. Architecture Design

The main architecture is divided into several layers of logic computation and memory block access, as shown in Fig. 3. The target FPGA is mapped with the related application. Some FPGA BRAM memory is reserved for basic operations.

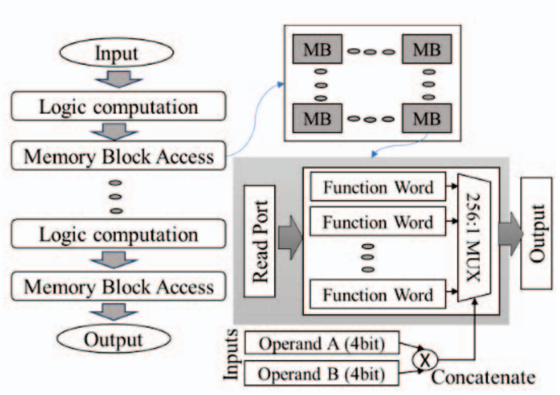

Fig. 3: Overall Structure

The available BRAM is virtually partitioned to several memory blocks (MBs). Each MB consists of 256 address spaces, and each address space comprises of a function word of 8 bits. These function words are essentially the computational results of the two 4 bit operands. The function words are read via the read port. If a different type of application needs to be loaded, the MBs can be re-loaded with a different data file via the read port. The MB is accessed via an 8 bit address line, which is essentially a combination of two operands of size 4 bit each. These also act as select lines for a 256:1 multiplexer, as shown in the Fig. 3. Based on the operand values, the correct function word is chosen. If the size of the inputs is more than 8 bits, then a series of MBs need to be accessed.

We use BRAM memory to store results that are associated with critical operations and associated with secret parameters. Other operations are executed normally in the LC layers.

B. Portraying Target Application via Data Flow Graph (DFG)

The target application (CNN for present case) is depicted as a DFG [23]. This DFG portrays the various computational operations as its nodes and the edges depict relation or data flow between the nodes.

C. Fusing the Nodes

To reduce runtime computations, parent and child nodes are fused, if there is no new inputs.

D. Ordering of Nodes

After all possible node fusions, computations in each node needs are mapped to one or more MBs. If available MBs are less than needed, the more power consuming operations are mapped first, followed by operations with less power consumption. So, nodes are ordered as per their power consumption, from maximum to minimum.

For CNN, nodes associated with multiple and accumulate (MAC) operations consume the most power, as well as associated with secret parameters, followed by tanh and finally element wise mathematical operations.

E. Mapping of Nodes to MBs

Nodes whose total input bits is equal to or less than 8 can be mapped to a single MB, else a series of MB access is required to map them. The designer determines the memory access pattern.

For instance, a MAC operation with two 8-bit inputs involves a series of MB access that comprises of various two 4-bit input multiplication and addition operations as depicted in Fig. 4. A single MB access is sufficient for tanh and sigmoid operations.

V. Mitigating Threat and Other Problems

A. Ensuring Security/ Prevent Leakage of Information

As discussed previously, weights and biases are the secret information that an adversary tries to steal. In our present case, we pre-compute the operations associated with the weights and biases and store them in the FPGA BRAMs. These are performed as follows:

1) Stealing during transit: As there is no movement of the weight and biases data from the memory to the processor, hence, these cannot be stolen during transit by an adversary. Moreover, the data that is stored in the FPGA BRAMs are pre-computed, and the number of computations merged is also not known to the adversary. Hence, getting back the pre-computed values from the data in transit is not possible for the adversary.

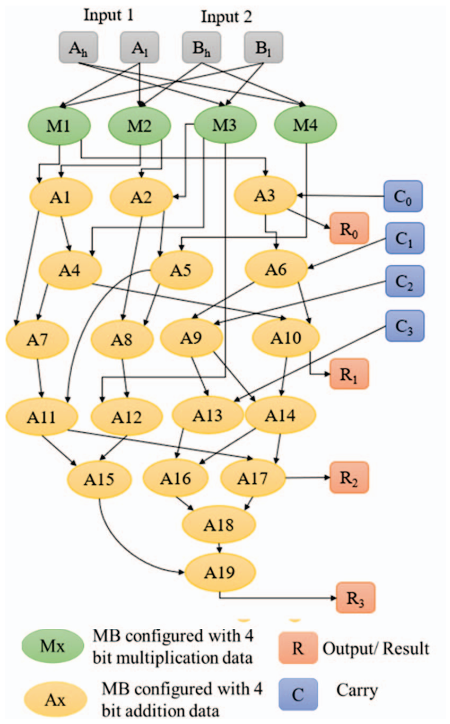

Fig. 4: Diagrammatic Representation of Design Flow

2) Stealing via side channel analysis: For the present case, no explicit operation is performed with the weights and biases. Only data access from the memory is performed, based on input operand for a particular computation. As power or timing is almost same for any type of data access from memory, the adversary will not be able to track the weights and bias from side channel analysis.

B. Resource, Power and Accuracy

In this mechanism, the available EMBs are utilized. Hence, the resource and power consumption is greatly reduced. Moreover, as pruning and optimization of the original design is not performed, hence, accuracy is not compromised.

C. Issue of Timing

In the proposed methodology, the time required for complex computations are significantly reduced via a look up memory operation. Thus, the operations completes quicker and more slack is obtained, which can be used for other operations. This improves throughput of the system and is quite beneficial for real time applications.

VI. Experimentation and Results

A. Experimentation Strategy

For experimentation, we adhere to the CNN design of [3], as the base design. Verilog codes are instantiated in Xilinx Vivado 2020.2 platform. This is targeted to ZynQ XCZU9EG Evaluation Board, operated at 150MHz clock frequency. After completing this base design, we determine the free EMBs and then modify the design as per our proposed mechanism and implement the modified design on the FPGA platform.

TABLE II: Comparison of Resource Utilization, Power and Energy Efficiency

Model	[6]	[7]	[2]	[3]	Proposed
Year	2018	2020	2020	2021	2023
Platform	Zynq VC7VX485T	Zynq XCZU9EG	Artix XC7A20	Zynq XCZU9EG	Zynq XCZU9EG
Clock (MHz)	-	150	50	100	150
Precision/ Bit	16	16	8	8	8
Power (W)	0.676	-	14.13	1.673	0.4387
Throughput (GOPs)	20.3	28.8	164.1	0.141	170.3
DSP	406	204	571	123	54
LUT	75,221	25,276	88,756	61,713	24,611
FF	38,577	66,569	42,038	27,863	11,869
Energy Efficiency (GOPs/W)	30.029	-	11.61	0.084	388.19

For the present work, we consider $m = n = 8$ and divide the available memory space into MBs that contain 256 function words, each of size 256 bits. These MBs can be accessed via a 8 bit address line.

B. Result Analysis

1) Comparison of results with other works: Table II provides a comparison of the current work with other existing works. The parameters of comparison are resource utilization, power, throughput and energy efficiency.

Resource utilization in the present case are digital signal processing (DSP) modules, look up tables (LUT), flip flops (FF) for the target FPGA platform. As evident from Table II, these parameters are significantly reduced, as most of the operations are precomputed and stored in the FPGA BRAMs, which reduces the load on the major computing FPGA resources, i.e. DSP, LUT and FFs.

Performance of the system is reflected by the throughput, which is represented by giga (billion) operations per second (GOPs). This depends on frequency of operation and per sample latency. As evident from Table II, throughput is quite high for the present work, as compared to other related works.

Power consumed by the system is measured in Watts (W). As evident from Table II, the power consumed by our mechanism is quite low than the related works. This is because most of the operations are precomputed and the reflected power is mainly associated with data access from FPGA BRAMs, which is quite low when compared to computations.

Energy efficiency is the ratio of throughput and power. This is also compared with existing works and shown in Table II. For our proposed mechanism, the power consumed is lower and throughput is higher than existing works, hence, energy efficiency is better than existing works.

2) Analysis with respect to availability of EMBs: Energy efficiency of a system is directly proportional to the throughput and inversely proportional to the power consumed by the system. These are directly linked with the available EMBs of a system. The following two analysis are performed as the total number of EMBs required to replace every functionality may not be available and we need to order the operations.

Throughput: Fig. 5 depicts the variation of throughput with respect to available EMBs. To ensure uniformaity during analysis, throughput is normalized, while the available EMBs is depicted as percentage availability. As evident from the graphical representation, the throughput increases steeply in the beginning and then stabilizes at the end. This is due to the fact that with mapping of more computational functions in the initial phases, more slack is available that are used to allocate other operations. However, when the more consuming functions are allocated, then low computational tasks are mapped, which does not affect the increase in throughput of the system and hence, it stabilizes at the end.

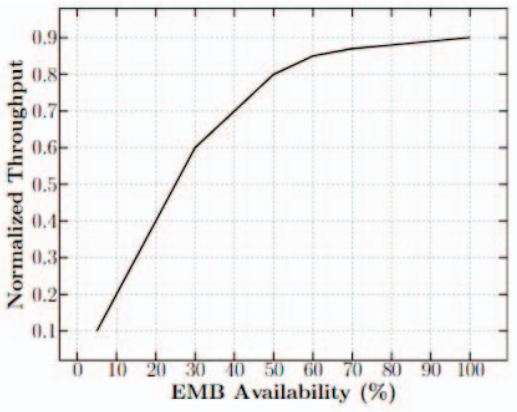

Fig. 5: Graphical analysis of throughput with respect to EMB availability

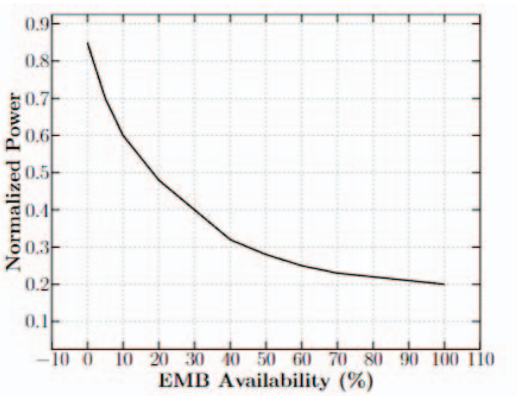

Fig. 6: Graphical analysis of power consumption with respect to EMB availability

Power: Fig. 6 depicts variation of power with respect to available EMBs. As expected, power consumption decreases with the increase in EMB availability. This is in a decreasing exponential type. Initially, more power consuming operations are mapped, hence, the decrease is steeper, which gradually slows down with mapping of less power consuming operations.

3) Ensuring Security: Fig. 7 provides a snapshot of the timing for different computational operations. When normal mechanisms or without memory lookup strategy is performed,

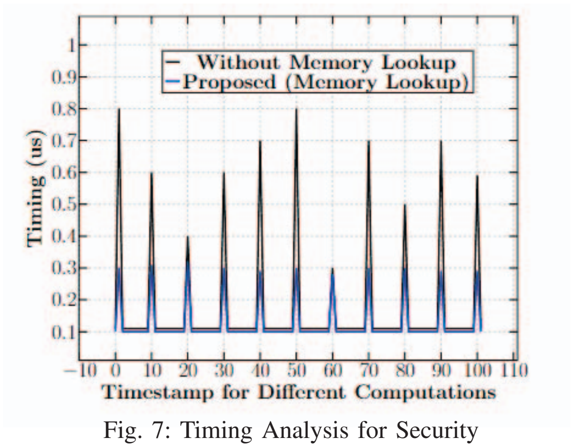

Fig. 7: Timing Analysis for Security

then via the timing analysis, an adversary may try to know the weights and biases used for the operation. This is prevented when our proposed mechanism is used or look up in memory computations are used. As these do not deviate significantly and remains almost the same, hence, an adversary cannot determine the secrets. Such analysis can also be performed with other side channel parameters like power, etc. However, in this paper, we limit to timing analysis.

VII. Conclusion

Neural network like CNN based applications are implemented on FPGAs for hardware acceleration. However, the computational intensive operations consume high resource and power. Existing strategies adhere to pruning that reduces accuracy. Moreover, they are also vulnerable to attacks from adversaries that tries to steal the secret weights and biases. We propose strategy where we pre-compute various operations along with the weights and biases in the pre-deployment phase and store them in FPGA memory. As evident from the experimental results, we achieve not only security, but also high accuracy, low resource utilization, low power consumption, accuracy, high throughput and high energy efficiency by our proposed mechanism. Presently, we consider only LeNet5 model, but this can be extended to different versions of CNN, which we plan to do in future. Alongside, we also plan to investigate the effect of precision and network size on EMBs to give a better insight into the effectiveness of this approach.

References

[1] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[2] D. Shan, G. Cong, and W. Lu, "A cnn accelerator on fpga with a flexible structure," in *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*. IEEE, 2020, pp. 211–216.

[3] 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, 2021.

[4] P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks," in *Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2007. Lecture Notes in Computer Science, vol 4599.*, 2007, pp. 443–453.

[5] A. Ghosh, S. Paul, J. Park, and S. Bhunia, "Improving energy efficiency in fpga through judicious mapping of computation to embedded memory blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 6, pp. 1314–1327, 2014.

[6] W. Chen, H. Wu, S. Wei, A. He, and H. Chen, "An asynchronous energy-efficient cnn accelerator with reconfigurable architecture," in *2018 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2018, pp. 51–54.

[7] Y. Shi, T. Gan, and S. Jiang, "Design of parallel acceleration method of convolutional neural network based on fpga," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2020, pp. 133–137.

[8] M. Méndez Real and R. Salvador, "Physical side-channel attacks on embedded neural networks: A survey," *Applied Sciences*, vol. 11, no. 15, p. 6790, 2021.

[9] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote fpga side-channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.

[12] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 workshops*. IEEE, 2011, pp. 109–116.

[13] Y.-H. Chen, T. Krishna *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[14] J. Li, Z. He, A. S. Rakin, D. Fan, and C. Chakrabarti, "Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 248–258.

[15] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[16] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[17] M. M. Ahmadi, L. Alrahis, A. Colucci, O. Sinanoglu, and M. Shafique, "Neurounlock: Unlocking the architecture of obfuscated deep neural networks," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 01–10.

[18] M. M. Ahmadi, L. Alrahis, O. Sinanoglu, and M. Shafique, "Dnn-alias: Deep neural network protection against side-channel attacks via layer balancing," *arXiv preprint arXiv:2303.06746*, 2023.

[19] W. Ligon, G. Monn, D. Stanzione, F. Stivers, and K. Underwood, "Implementation and analysis of numerical components for reconfigurable computing," in *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403)*, vol. 2, 1999, pp. 325–335 vol.2.

[20] C. Brunelli, H. Berg, and D. Guevorkian, "Approximating sine functions using variable-precision taylor polynomials," in *2009 IEEE Workshop on Signal Processing Systems*, 2009, pp. 057–062.

[21] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. Prajapati, M. A. Indovina, S. M. P. Dinakarrao, and A. Ganguly, "Look-up-table based processing-in-memory architecture with programmable precision-scaling for deep learning applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 263–275, 2022.

[22] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Transactions on Computers*, vol. 54, pp. 319–330, 2005.

[23] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque, "Hw2vec: a graph learning tool for automating hardware security," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 13–23.